Złożenie pracy online:
2021-03-03 10:26:42
Kod pracy:
7551/37697/CloudA

Jacek Kopka
(nr albumu: 23279)

Praca inżynierska

"ITDesk" - aplikacja internetowa do prowadzenia ewidencji zgłoszeń i czasu pracy pracownika

"ITDesk" - Web application for keeping records of notifications and employee working time

Wydział: Wydział Nauk Społecznych i Informatyki

Kierunek: Informatyka

Specjalność: programowanie aplikacji biznesowych

Promotor: dr Krzysztof Przybycień

Nowy Sacz, 2021

Składam najserdeczniejsze podziękowania dla Promotora mojej pracy, doktorowi Krzysztofowi Przybycieniowi, za wskazówki udzielone mi podczas pisania, cenne uwagi i merytoryczne porady, a także za cały poświęcony mi czas.

Streszczenie

Aplikacja internetowa przeznaczona dla działu Helpdesk umożliwia tworzenie, realizację i ewidencjonowanie zgłoszeń dla klientów posiadających konto w systemie. Projekt zrealizowany w oparciu o najnowsze technologie, utworzony w środowisku programistycznym Visual Studio wykorzystując platformę ASP.NET Core. Funkcjonalność systemu pozwala na utworzenie kont dla użytkowników z podziałem na role i przypisywaniem uprawnień dostępu do wybranych modułów. Ewidencja zadań i zgłoszeń zapewnia dostęp do historii prowadzonych prac, a moduł raportowania z szerokim wachlarzem dostępnych filtrów wspomaga analizę zebranych danych.

Słowa kluczowe

ASP.NET Core, .NET Core MVC, Entity Framework Core, ASP.NET Core Identity, Bootstrap, Aplikacja internetowa, Helpdesk

Abstract

Web application especially for the Helpdesk, enables the creation, implementation and recording of notifications in the system.The project is based on the latest technologies, created in the Visual Studio development environment using the ASP.NET Core platform.All features of the system allows you to create accounts for users with division into roles and assigning access rights to selected modules.The record of tasks and notifications provide access to your work history, and a reporting module with a wide range of filters available, helps you analyze the collected data.

Keywords

ASP.NET Core, .NET Core MVC, Entity Framework Core, ASP.NET Core Identity, Bootstrap, Web application, Helpdesk

Spis treści

1. Wstęp

1.1 Cel i zakres prac

Celem zintegrowanej aplikacji internetowej ITDesk jest ewidencjonowanie zgłoszeń, które ma służyć usprawnieniu pracy działu Helpdesk w branży informatycznej z zakresu pełnienia pomocy i wsparcia technicznego dla klienta końcowego. System zapewnia dostęp do bazy danych użytkowników aplikacji, klientów i prowadzonych zgłoszeń. Pozwala na gromadzenie, przetwarzanie i raportowanie czasu pracy pracownika spędzonego nad zgłoszeniem.

1.2 Założenia wstępne

W ramach podstawowych ustaleń system dzieli się na dwa osobne moduły “ITDesk-Portal” oraz “ITDesk-Intranet”. Analogicznie pierwszy dla klienta końcowego dzięki której ma przejrzysty i szybki wgląd do swoich zgłoszeń oraz drugi dostępny dla pracowników firmy świadczącej usługi z możliwością pełnej obsługi zgłoszenia. Odpowiednio do obu systemów użytkownik może uzyskać dostęp z dowolnego stanowiska wyposażonego w łącze i przeglądarkę internetową. Podczas projektowania aplikacji wzięto pod uwagę responsywność stron WWW zapewniając pełną skalowność dla urządzeń z którymi użytkownik może wchodzić w interakcję takie jak: komputer stacjonarny, laptop, tablet lub smartphone.

Wychodząc naprzeciw oczekiwaniom klientów obie aplikacje zostały stworzone z wykorzystaniem najnowszych technologii ASP.NET zapewniając bezpieczeństwo przechowywania oraz przetwarzania danych, pełną konfigurację wraz z uprawnieniami dostępu dla wybranych użytkowników, widoków opartych o m.in. framework Bootstrap dostosowanych do obecnych standardów przeglądarek internetowych.

Głównym założeniem aplikacji była możliwość przydzielania zadań wybranym pracownikom firmy oraz ewidencjonowania czasu i postępu prac spędzonego nad zgłoszeniami. Atutem po stronie klienta jak i pracownika jest możliwość dostępu do sekcji “Raporty” w której w łatwy sposób można przeglądać status wykonywanych zgłoszeń i porównywać wedle wskazanych parametrów.

2. Architektura i wykorzystane technologie

Ogólna koncepcja systemu opiera się na dwóch oddzielnych aplikacjach działających współbieżnie z dostępem do wspólnej bazy danych, gdzie przechowywane są wszelkie informacje na temat użytkowników, prowadzonych zgłoszeń i czasu pracy pracownika.

2.1 Platforma ASP.NET Core

Obie aplikacje internetowe zostały zaprojektowane opierając się o najnowsze wytyczne ASP.NET Core w wersji 3.1 dostarczanych przez firmę Microsoft. Jest to rozbudowana międzyplatformowa platforma typu "Open Source", która[1]:

- umożliwia tworzenie nowoczesnych aplikacji sieci Web i interfejsów API,
- używanie narzędzi programistycznych w systemach Windows, macOS i Linux,
- pozwala na wdrażanie aplikacji w chmurze lub lokalnie
- uruchamiana na platformie .NET Core

2.1.1 Wzorzec MVC[2]

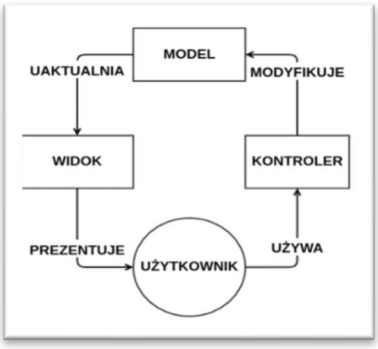
Wzorzec architektoniczny Model-View-Controller (MVC) pozwala oddzielić aplikację do trzech głównych grup składników[3]:

- Modeli
- Widoków
- Kontrolerów

Wszystkie komponenty są ze sobą wspólnie połączone. Korzystając z tego wzorca, zapytania i interakcje użytkownika na widoku są kierowane do kontrolera, który przyjmuje dane wejściowe i jest odpowiedzialny za pracę z modelem w celu wykonywania akcji. Kontroler wybiera widok, który ma być udostępniony dla użytkownika i przesyła oczekiwane dane modelu.

[1] https://docs.microsoft.com/pl-pl/aspnet/core/?view=asp netcore-5.0 (data odczytu 15.02.2021)
[2] Adam Freeman, Pro ASP.NET Core MVC: Sixth Edition, Apress 2016 s. 6
[3] Tamże, s. 53

Rysunek 1. Model MVC



"Zmiana danych przez użytkownika odbywa się za pomocą kontrolera, ten modyfikuje model, model odświeża widok, a informacja z widoku dociera do użytkownika."[4]

Role związane z modelem

Model w aplikacji MVC reprezentuje stan aplikacji i wszelkie operacje logiki biznesowej, które powinny być wykonywane przez nią. Logika powinna być odizolowana w modelu razem z dowolnymi metodami dokonującymi zmian w aplikacji.

Wyświetlanie widoków

Widoki są odpowiedzialne za przedstawienie danych za pomocą graficznego interfejsu użytkownika. Mogą być podzielone na mniejsze podwidoki składające się na całość wyświetlanego widoku.

Obowiązki kontrolera

Na zadania kontrolera składa się odbieranie, przetwarzanie oraz analiza danych od użytkownika. W jednej chwili tylko jeden kontroler może sterować aplikacją zmieniając stan modelu, odświeżając widok lub przełączyć sterowanie na inny kontroler. W aplikacjach MVC widok prezentuje tylko dane; kontroler obsługuje je i reaguje na interakcję.

2.1.2 C#

Jeden z najmłodszych języków programowania opracowany dla firmy Microsoft przez Andersa Hejlsberg'a. Obiektowy, wysokopoziomowy i bezpieczny dla typu, który ma wiele wspólnych cech z językami C++ oraz Java[5].

[4] Źródło pl.wikipedia.org (https://pl.wikipedia.org/wiki/Model-View-Controller) (data odczytu 15.02.2021)
[5] Adam Boduch, „Wstęp do programowania w języku C#", Helion 2006, s.14

Używany do tworzenia aplikacji za pomocą frameworka ASP.NET, typu open-source, posiada bardzo rozbudowaną bibliotekę klas BCL do rozwijania aplikacji konsolowych, okienkowych oraz sieciowych.

Kod źródłowy zapisany w języku C# jest następnie kompilowany do języka pośredniego (IL lub CIL Common Language Infrastructure) – języka najniższego poziomu platformy .NET i tam wykonywany. Uruchomienie w systemie operacyjnym programu napisanego w języku c# bez kompilacji w odpowiednim środowisku jest niemożliwe.[6]

Podczas tworzenia aplikacji w środowisku Visual Studio 2019 wykorzystano najnowszą wersję języka C# 8.0 przeznaczoną dla platformy .NET Core.

2.1.3 ASP.NET Core Identity

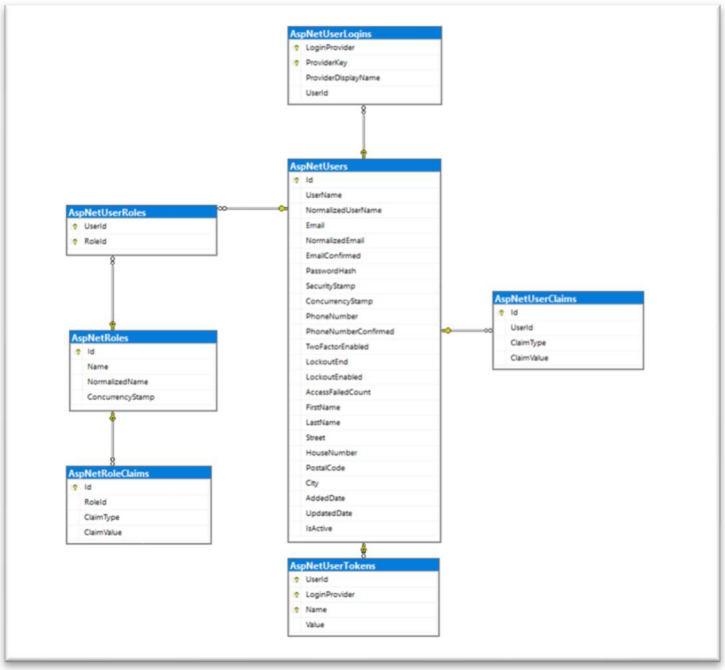
Interfejs API ściśle związany z ASP.NET Core dostarcza bliblioteki klas obsługujących funkcje logowania użytkownika.

Każdy użytkownik może stworzyć konto z danymi logowania przechowywanymi w usłudze Identity lub skorzystać z zewnętrznych loginów dostarczanych przez serwis Facebook, Google, konto Microsoft oraz Twitter. Zazwyczaj konfiguracja odbywa się za pomocą SQL Server do przechowywania danych użytkowników, takich jak: nazw profili, adresów email, haseł oraz danych osobowych. Istnieje możliwość użycia innego magazynu danych np. Azure Table Storage.[7]

W systemie wykorzystano interfejs w wersji 2.2.3, struktura tabel w bazie danych prezentuje się następująco.

[6] https://pl.wikipedia.org/wiki/C_Sharp (data odczytu 15.02.2021)
[7] https://docs.microsoft.com/pl-pl/aspnet/core/security/authentication/identity?view=aspnetcore-5.0&tabs=visual-studio (data odczytu 15.02.2021)

Rysunek 2. Struktura ASP.NET Core Identity w SQL Server



2.1.3.1 ASP.NET Core Authorization

Autoryzacja proces identyfikacji użytkownika w systemie, określa co użytkownik może, a czego nie może zrobić. Np. jeśli użytkownik jest zalogowany jako Administrator ma dostęp do wszystkich funkcji aplikacji, przy czym zwykły użytkownik ma ograniczone uprawnienia. Oparta na rolach i oświadczeniach, w skład autoryzacji wchodzi co najmniej jedno wymaganie.

2.1.3.2 ASP.NET Core Authentication

Uwierzytelnianie jako proces ustalania poświadczeń użytkownika. Zapewnia identyfikację czy dany użytkownik posiada dostęp do określonych funkcji systemu.

2.2 Microsoft SQL Server 2017

Podczas tworzenia „ITDesk" za trwałe magazynowanie i ścisłą współpracę wymiany danych z aplikacjami odpowiadał SQL Server 2017 w pełnej wersji. Odpowiedzialny za gromadzenie, przechowywanie i prezentację wymaganych informacji system zarządzania relacyjnymi bazami danych został stworzonym i jest wspieranym przez firmę Microsoft.

Pojęciem SQL (ang. Structured Query Language) oznaczamy język zapytań używany do wykonywania operacji na bazach danych, rozszerzeniem tego języka jest Transact-SQL (T-SQL) przeznaczony dla interakcji z serwerami baz danych wcześniej Sybase – firmy produkującej systemy bazodanowe, obecnie prawa do wykorzystywania tego języka kupiła firma Microsoft i sukcesywnie wprowadza do swoich kolejnych produktów MS SQL Server.

Rozwój standardowego języka SQL o T-SQL inaczej transakcyjny SQL pozwala na tworzenie procedur i wyzwalaczy oraz korzystanie z takich składni jak pętle, zmienne oraz instrukcje warunkowe w zapytaniach do bazy danych.[8]

SQL Server jest ściśle powiązany i jest integralną częścią systemu „ITDesk", praca bez bazy danych i odpowiedniego silnika nie jest możliwa. Wykorzystanie przedstawionej wersji SQL Server nie jest wymagane lecz ze względów technicznych zalecana jest taka sama lub nowsza odsłona SQL Server wraz z dystrybucją Express, która posiada pewne ograniczenia w wykorzystywaniu sprzętu jak i Azure SQL – aparatu bazy danych jako usługi w chmurze.

2.3 Entity Framework Core

Technologia pozwalająca wykonywać operacje na bazach danych z zastosowaniem dwóch głównych sposobów:

- Code-First metoda, w której w pierwszej kolejności powstają klasy C#, odpowiadające każdej tabeli w bazie danych oraz jedna klasa główna, która odpowiada całej bazie. Następnie przy pomocy Entity Framework na podstawie napisanych klas tworzona jest automatycznie baza danych.
- Database-First drugi sposób, w którym to na bazie danych która została utworzona w pierwszej kolejności na serwerze bazodanowym są automatycznie tworzone klasy odpowiadające poszczególnym tabelom. Każdej tabeli odpowiada jedna klasa oraz klasa wspólna dla wszystkich tabel.

Aplikacje są oparte na wersji Entity Framework Core 3.1.5 wykorzystują podejście Code-First, wszystkie klasy wspólne dla obu modułów obsługuje warstwa „ITDesk.Data", która odpowiada za wymianę informacji z bazą danych.

[8] https://pl.wikipedia.org/wiki/Transact-SQL (data odczytu 16.02.2021)

Rysunek 3. Fragment klasy odpowiadającej za tworzenie tabeli „Request”

```
[Key]
[DatabaseGenerated(DatabaseGeneratedOption.Identity)]
public string Id { get; set; }

[Required(ErrorMessage = "Klient jest wymagany")]
public string ClientId { get; set; }

[Required(ErrorMessage = "Nazwa jest wymagana")]
[Column(TypeName = "nvarchar(124)")]
public string Name { get; set; }

[Required(ErrorMessage = "Opis jest wymagany")]
public string Description { get; set; }
public string RequestNumber { get; set; }
[DatabaseGenerated(DatabaseGeneratedOption.Identity)]
public int Counter { get; set; }
```

2.3.1 LINQ

Działając w oparciu o język SQL technologia LINQ (**Language INtegrated Query**) pozwala nam na wykonywanie zapytań do obiektów. Podobnie jak w przypadku zapytań do bazy danych w SQL, LINQ wykorzystuje standardowe operatory **SELECT**, **FROM**, **WHERE** jak i rozszerzenia ich o funkcje **DISTINCT(), SUM(), ORDER BY()** do zwracania pewnej części danych dostosowanych do naszych potrzeb.

2.3.2 Wyrażenie lambda

W zapytaniach LINQ istnieje możliwość tworzenia delegatów jako anonimowej metody inaczej wyrażenia lambda, które pozwala użyć jej bez deklaracji. Przykład takiego wyrażenia poniżej. [9]

Rysunek 4. Fragment kodu pobierającego listę użytkowników

```
var listUser = _context.Users
    .Join(_context.UserRoles, u => u.Id, r => r.UserId, (user, role) => new { user, role })
    .Join(_context.Roles, x => x.role.RoleId, y => y.Id, (role1, roleId) => new { role1, roleId })
    .Where(z => z.roleId.NormalizedName != "KLIENT")
    .Where(z => z.role1.user.IsActive == true)
    .Select(m => m.role1.user).Distinct();
```

[9] https://www.plukasiewicz.net/Artykuly/Wyrazenia_lambda (data odczytu 16.02.2021)

2.4 Simple Mail Transfer Protokol – obsługa poczty

W skrócie **SMTP**, relatywnie prosty protokół realizujący sposób przekazywania poczty elektronicznej pomiędzy nadawcą i odbiorcą, który określa przynajmniej jednego adresata wiadomości i przekazuje zawartość wiadomość. W aplikacji za obsługę poczty odpowiada serwer SMTP Gmail, który pozwala na bezpłatne wysłanie do 2000 wiadomości e-mail dziennie na które składają się powiadomienia użytkownika o wykonywanych działaniach na jego koncie. Podstawowa konfiguracja to:

- Identyfikator Gmail – adres, który będzie pełnił rolę nadawcy wiadomości,
- Hasło do poczty,
- Adres hosta lub serwera SMTP – w tym wypadku **smtp.gmail.com**
- Port – 465 (komunikacja SSL) lub 587 (komunikacja TLS)

2.5 HTML i CSS

HyperText Markup Language w skrócie HTML, kod, który odpowiada za stworzenie struktury, tego jak strona ma być prezentowana i wyświetlana wraz z zawartością.

Uzupełnieniem tego języka do nadawania bardziej szczegółowych formatowań dla wizualizacji strony i dostosowania do wymagań użytkowników wykorzystywane są kaskadowe arkusze stylów inaczej CSS.

Połączenie tych dwóch języków jest idealnym rozwiązaniem dla prezentowania bardziej nowoczesnych i atrakcyjnych wyglądowo stron WWW, zgodnych z najnowszymi standardami.

2.5.1 Bootstrap v4

W oparciu o powyższe języki łącząc wszelkie rozwiązania i możliwości ich obu powstała biblioteka Bootstrap za którą odpowiadają deweloperzy z Twittera, zawierająca gotowe mechanizmy do wykorzystania podczas tworzenia interfejsów graficznych stron i aplikacji. Główną zaletą użycia frameworaka Bootstrap jest prostota implementacji oraz konfiguracji według swoich potrzeb i założeń projektowych. [10]

W aplikacjach „ITDesk-Portal” i „ITDesk-Intranet” budowane według wstępnie ustalonego szablonu wykorzystano Bootstrap w wersji v.4.3.1

[10] https://pl.wikipedia.org/wiki/Bootstrap_(framework) (data odczytu 17.02.2021)

2.5.2 JavaScript

Chcą zapewnić jakąkolwiek interakcję wizualną dla użytkownika reagując na wszelkie zdarzenia najczęściej spotykanym uzupełnieniem HTML i CSS jest język skryptowy JavaScript. Pozwala na zastosowanie animacji prowadząc użytkownika po stronie za pomocą efektów wizualnych, jak i walidację na wprowadzanych danych.[11]

2.5.3 jQuery

Podobnie jak w przypadku Bootstrap w HTML i CSS tak w języku JavaScript posiadamy bibliotekę jQuery z dostępem do najpopularniejszych mechanizmów ułatwiając korzystanie z języka. Pozwala na przyspieszenie i uproszczenie pracy programisty oferując gotowe funkcje do wykorzystania. Duża popularność tej biblioteki zaowocowała stworzeniem przez społeczność dodatków (tzw. wtyczek jQuery) oferujących zawansowane efekty wizualne, galerie, przejścia.[12]

2.5.4 DataTabels

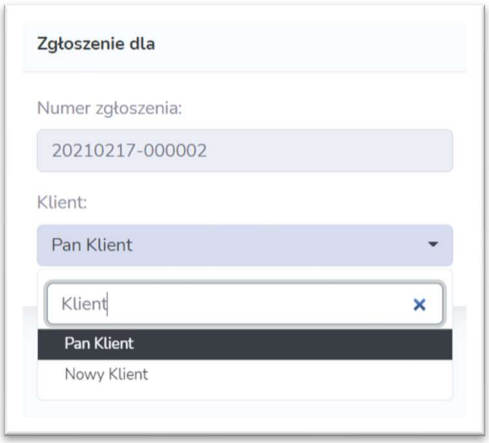
Projekt „ITDesk" bazuje na stworzonych danych przez użytkowników systemu i prezentuje je w większości przypadków za pomocą tabeli, która pozwala na szybki i przejrzysty dostęp do dużej części zebranych informacji. W obu modułach została zastosowana gotowa wtyczka jQuery DataTables v1.10.23 zapewniając kontrolę z interakcją na tabeli, dla developerów daje elastyczne metody konfiguracji, to w jaki sposób i jak mają być prezentowane informacje dla klienta końcowego.

2.5.5 Bootstrap select

Kolejna wtyczka jQuery w wersji v1.13.14 zaimplementowana w projekcie, używając dostępnej opcji **data-live-search="true"** wzbogaca wyświetlanie okna wejściowego **<select> </select>** o moduł wyszukiwania danych zawartych w liście rozwijanej w czasie rzeczywistym bez przeładowywania strony.[13]

[11] D. S. McFaeland, „Javascript i jQuery nieoficjalny podręcznik", Helion 2012, s.18-19
[12] Tamże, s. 17-18
[13] https://developer.snapappointments.com/bootstrap-select/ (data odczytu 17.02.2021)

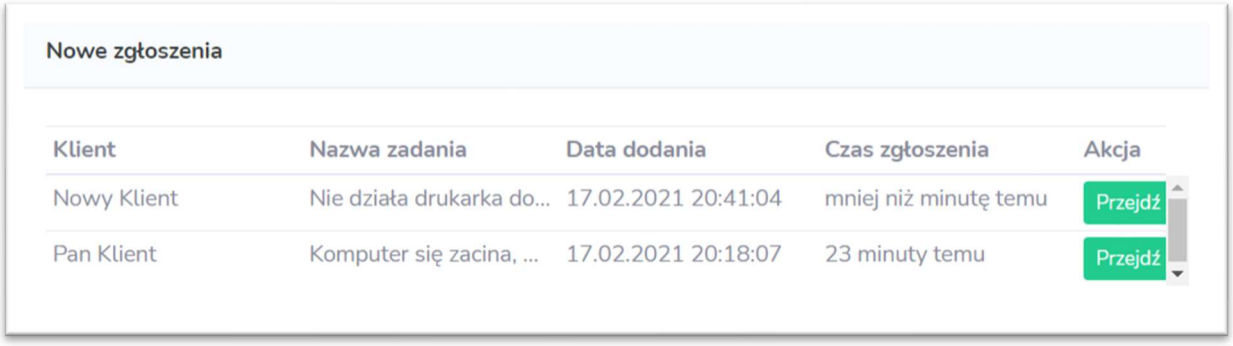
Rysunek 5. Przykład selectora Bootstrap



2.5.6 TimeAago

Następnym dodatkiem uzupełniającym wyświetlanie i prezentowanie w bardziej przejrzystej formie informacji o czasie jest TimeAgo. Gotowa biblioteka jQuery jako dane wejściowe otrzymuje czas w formacie ISO 8601 np. 2004-05-23T14:25:10.487 i porównując ją z aktualnym czasem serwera aplikacji prezentuje użytkownikowi ile czasu upłynęło pomiędzy tymi dwoma datami.[14]

Rysunek 6. Wykorzystanie TimeAgo w praktyce



2.6 Microsoft Azure

Platforma stworzona prze firmę Microsoft udostępnia poszczególne zasoby będące w chmurze, jak serwery Azure SQL, maszyny wirtualne przeznaczone do wykonywania obliczeń, prowadzenia testów aplikacji i programów. Pozwala na umieszczenie aplikacji na zdalnych serwerach i dostęp do statystyk użycia systemu, zapewnia bezpieczeństwo przechowywanych danych korzystając z replikacji wdrożonych projektów umieszczając poszczególne kopie w

[14] https://timeago.org/ (data odczytu 17.02.2021)

różnych miejscach obliczeniowych, eliminując tutaj brak dostępu do danych w przypadku fizycznej awarii komputera.

2.7 Platforma programowo-sprzętowa

Proces tworzenia projektu wspomagały dwa środowiska, Visual Studio 2019 oraz Microsoft SQL Server Managment Studio 17.

2.7.1 Visual Studio 2019

Całość aplikacji powstała przy pomocy środowiska programistycznego Visual Studio w najnowszej dostępnej odsłonie 2019 zapewniając wsparcie dla C# 8.0 oraz .NET Core 3.0. Posiada wbudowany edytor kodu wspierający automatyczne uzupełnianie treści pisanych przez programistę – IntelliSense, podpowiadając dostępne opcje z których może skorzystać podczas tworzenia kodu. [15]

2.7.2 Microsoft SQL Server Managment 17

Za pomoc w konfiguracji silnika bazy danych, komunikacji i administracją danymi bezpośrednio w bazie danych wspomagając proces budowy i testowania aplikacji odpowiada środowisko wchodzące w skład Microsoft SQL Server. Jako narzędzie do zarządzana, umożliwia tworzenie zapytań, procedur i przeglądanie danych.

[15] https://pl.wikipedia.org/wiki/Microsoft_Visual_Studio (data odczytu 17.02.2021)

3. Zakres i procedura instalacji

3.1 Model założeniowy

Ustalonym założeniem projektu było rozdzielenie dostępu dla użytkowników dla poszczególnych aplikacji. Na stronie „ITDesk-Portal" klient tworząc konto uzyskuje dostęp tylko i wyłącznie do zasobów oferowanych w obrębie tego środowiska. Nie jest możliwe zalogowanie się użytkownika z portalu klienta w aplikacji przeznaczonej dla pracownika i na odwrót. Po stronie intranetu domyślnym, wbudowanym kontem jest konto **SuperAdmin** posiadające największe uprawnienia. Utworzenie nowego dostępu dla osoby w tej aplikacji odbywa się z poziomu konta administracyjnego wewnątrz systemu.

Obie aplikacje opierają swoje działanie na 4 podstawowych rolach: Klient, Pracownik, Administrator, SuperAdmin odpowiednio od najmniejszych do największych uprawnień. Więcej informacji dotyczących uprawnień dostępu dla poszczególnych ról opisano w sekcji „Bezpieczeństwo" -> „Role i uprawnienia".

3.2 Pierwsze uruchomienie, konfiguracja

Podczas pierwszego uruchomienia aplikacji automatycznie jest tworzona baza danych z odpowiednimi tabelami wymagana do poprawnego działania programu, ważne jest więc uprzednie skonfigurowanie połączenia do silnika bazodanowego. Uruchomienie aplikacji pomijając ten krok zwróci błąd.

System chodź składa się z dwóch osobnych aplikacji korzysta z tej samej bazy danych więc wymagane jest utworzenie połączenia w dwóch plikach w takiej samej konfiguracji odpowiednio dla jednej i drugiej aplikacji.

Po uruchomieniu solucji i załadowaniu wstępnych danych w programie Visual Studio przechodzimy do pliku „appsettings.json" znajdującym się w głównym folderze projektu, zmiany należy dokonać w sekcji „ConnectionString" modyfikując wartość po słowie kluczowym **Server=** przypisując adres silnika baz danych. Domyślnie aplikacja skonfigurowana jest z serwerem na adresie **127.0.0.1**, port **1433.**

Rysunek 7. Przykład konfiguracji połączenia do bazy danych

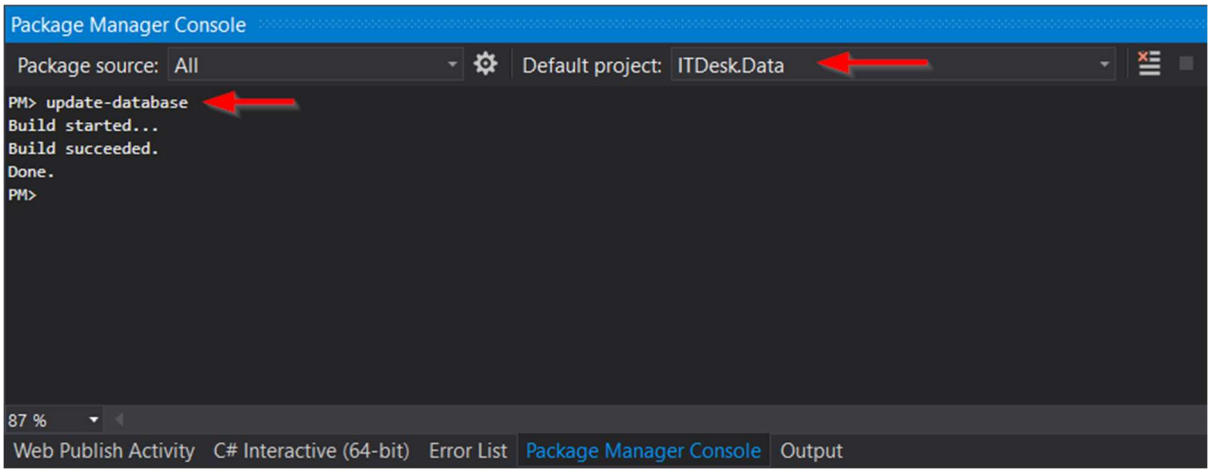
```
"ConnectionStrings": {
    "ITDeskContext": "Server=127.0.0.1,1433;Database=ITDesk;Trusted_Connection=True;MultipleActiveResultSets=true"
  }
```

Analogicznie postępujemy w przypadku drugiego projektu ustawiając takie same parametry.

Parametr **Database** wskazuje na nazwę bazy danych, która zostanie utworzona. Podczas pierwszej konfiguracji istnieje możliwość zmiany tej wartości, jednak zaleca się pozostawienie jej w niezmienionej formie.

Następnie w celu utworzenia bazy danych przechodząc do Package Manager Console w głównym oknie programu wpisujemy polecenie **update-database** i wybieramy domyślny projekt **ITDesk.Data**

Rysunek 8. PMC i pierwsze tworzenie bazy danych



Pomyślne zakończenie operacji zwróci powyższe komunikaty tworząc bazę danych z domyślnymi wartościami, które zostały zainicjonowane w kodzie. Uruchomienie aplikacji „wstrzykuje" do bazy wymagane dane do poprawnego działania.

4. Bezpieczeństwo

Podstawowym i wbudowanym kontem jest konto o największych uprawnieniach **SuperAdmin.**

Pierwsze uruchomienie strony Intranet automatycznie inicjuje utworzenie takiego konta o podanych danych:

- Login: itdesk.project@gmail.com
- Hasło: zaq1@WSX
- Rola: SuperAdmin

4.1 Role i uprawnienia, kontrola dostępu

Podział użytkowników aplikacji definiując ich przydział i nadając dostęp do poszczególnych wewnętrznych modułów zapewnia zaimplementowanie hierarchii opartej na rolach i uprawnieniach. Wyróżnia się 4 typy użytkownika idąc kolejno od najmniejszych uprawnień:

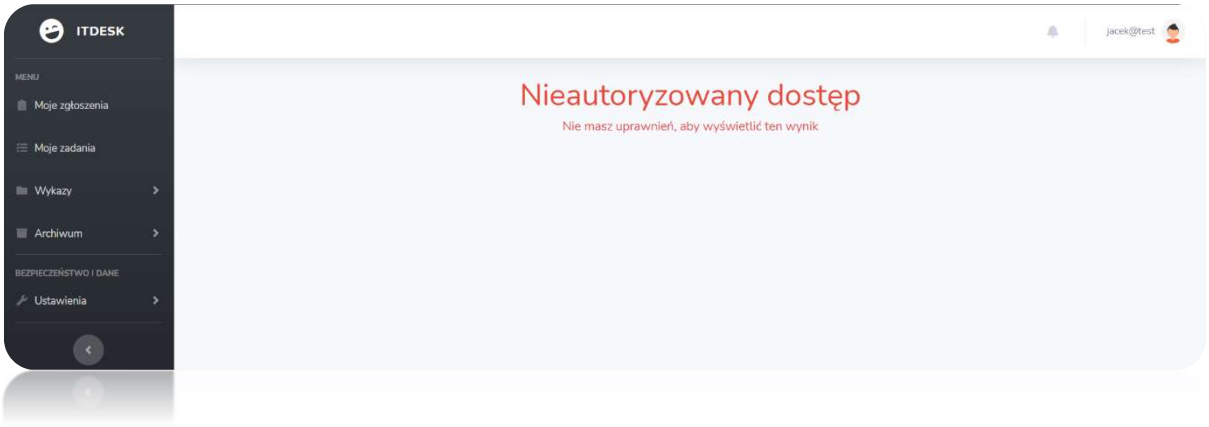
- Klient,
- Pracownik,
- Administrator,
- SuperAdmin

Kontrolę i opcje dostępu do wybranych funkcji zapewnia opisany wcześniej interfejs ASP.NET Identity.

Największe możliwości posiada użytkownik przypisany do roli SuperAdmin ignorując wszelkie inne uprawnienia zarządzania. W takim wypadku nie są brane pod uwagę wartości, to w jakiej roli znajduje się użytkownik i czy posiada dostęp do danej funkcji. Wprowadzenie takiej roli zabezpiecza przed możliwością usunięcia wszystkich administratorów z systemu.

Samo przypisanie użytkownika do roli zdefiniuje tylko jego typ (pomijając powyższy przypadek roli SuperAdmin), ważne jest też odpowiednie nadanie uprawnień i możliwości wykonywania zdefiniowanych funkcji. Odebranie określonej funkcji dezaktywuje dostęp nie tylko do wykonania akcji w aplikacji, ale również wyłączy graficzny przycisk do przekierowania na interfejsie. Próby uzyskania dostępu do wybranej funkcjonalności, która jest odebrana użytkownikowi zwróci stosowny komunikat.

Rysunek 9. Odmowa dostępu - przykład



Najmniejsze możliwości konfiguracji posiada rola Klient w najniższej konfiguracji pozostawiając możliwość wyłącznie tworzenia i przeglądania swoich zgłoszeń poprzez modyfikację swojego profilu i przeglądanie raportów.

Występuje tutaj swego rodzaju dziedziczenie w uprawnieniach, każdy typ o większych uprawnieniach ma to co niżej w hierarchii i więcej.

Rysunek 10. Lista uprawnień w systemie z podziałem na role - opracowanie własne.

Uprawnienie/Rola	Klient	Pracownik	Administrator	SuperAdmin
Zarządzanie klientami	✔	✔	✔	✔
Przeglądanie raportów	✔	✔	✔	✔
Tworzenie zgłoszenia	✔	✔	✔	✔
Zarządzanie zgłoszeniami	✔	✔	✔	✔
Usuwanie zadania		✔	✔	✔
Edycja zadania		✔	✔	✔
Tworzenie zadania		✔	✔	✔
Usuwanie zgłoszenia		✔	✔	✔
Edycja zgłoszenia		✔	✔	✔
Zarządzanie klientami na użytkowniku			✔	✔
Zarządzanie usuniętymi użytkownikami			✔	✔
Zarządzanie użytkownikami			✔	✔
Usuwanie użytkownika			✔	✔

Edycja użytkownika			✔	✔
Tworzenie użytkownika			✔	✔
Zarządzanie użytkownikami na kliencie			✔	✔
Zarządzanie usuniętymi klientami			✔	✔
Usuwanie klienta			✔	✔
Edycja klienta			✔	✔
Tworzenie klienta			✔	✔
Zarządzanie rolami na użytkowniku			✔	✔
Zarządzanie uprawnieniami na użytkowniku			✔	✔
Zarządzanie usuniętymi zadaniami			✔	✔
Zarządzanie usuniętymi zgłoszeniami			✔	✔
Zarządzanie zadaniami			✔	✔
Potwierdzanie konta				✔
Zarządzanie użytkownikami w roli				✔
Usuwanie roli				✔
Edycja roli				✔
Tworzenie roli				✔

4.2 Zabezpieczenia dodatkowe

Zastosowanie ASP.NET Identity pozwoliło ograniczyć dostęp i kontrolę nad kontem dla nieautoryzowanych osób. Zaimplementowanie w aplikacji protokołu SMTP dodatkowo zwiększając bezpieczeństwo, oprócz standardowych powiadomień dotyczących zgłoszeń wykorzystano również do wysyłania informacji na temat zmian dotyczących haseł i konta. Podczas rejestracji użytkownika w systemie pierwsze zalogowanie wymaga aktywacji konta, link aktywacyjny wysyłany jest na podany wcześniej adres email. System daje czas życia dla linku aktywacyjnego do 3 dni, po tym czasie link jest nieaktywny, a sposób aktywacji konta należy zgłosić do administratora systemu. W przypadku zmiany hasła wysyłana jest stosowna informacja, w linku przekierowującym zaszyty jest specjalny token, również i w tym wypadku czas jaki użytkownik posiada na dokonanie zmian wynosi do 3 godzin.

4.3 Hasła

Domyślnie przyjętą polityką haseł w aplikacji jest odpowiedni jego standard, każde hasło musi:

- Składać się z co najmniej z 8 znaków,
- Posiadać przynajmniej jeden znak alfanumeryczny (litery, cyfry, znaki interpunkcyjne),
- Posiadać przynajmniej jedną małą literę (‘a’ – ‘z’),
- Posiadać przynajmniej jedną wielką literę (‘A’ – ‘Z’),
- Posiadać przynajmniej jeden znak specjalny,
- Wymaga podania liczby z zakresu 0-9,
- Wymaga podania 2 odrębnych znaków w haśle

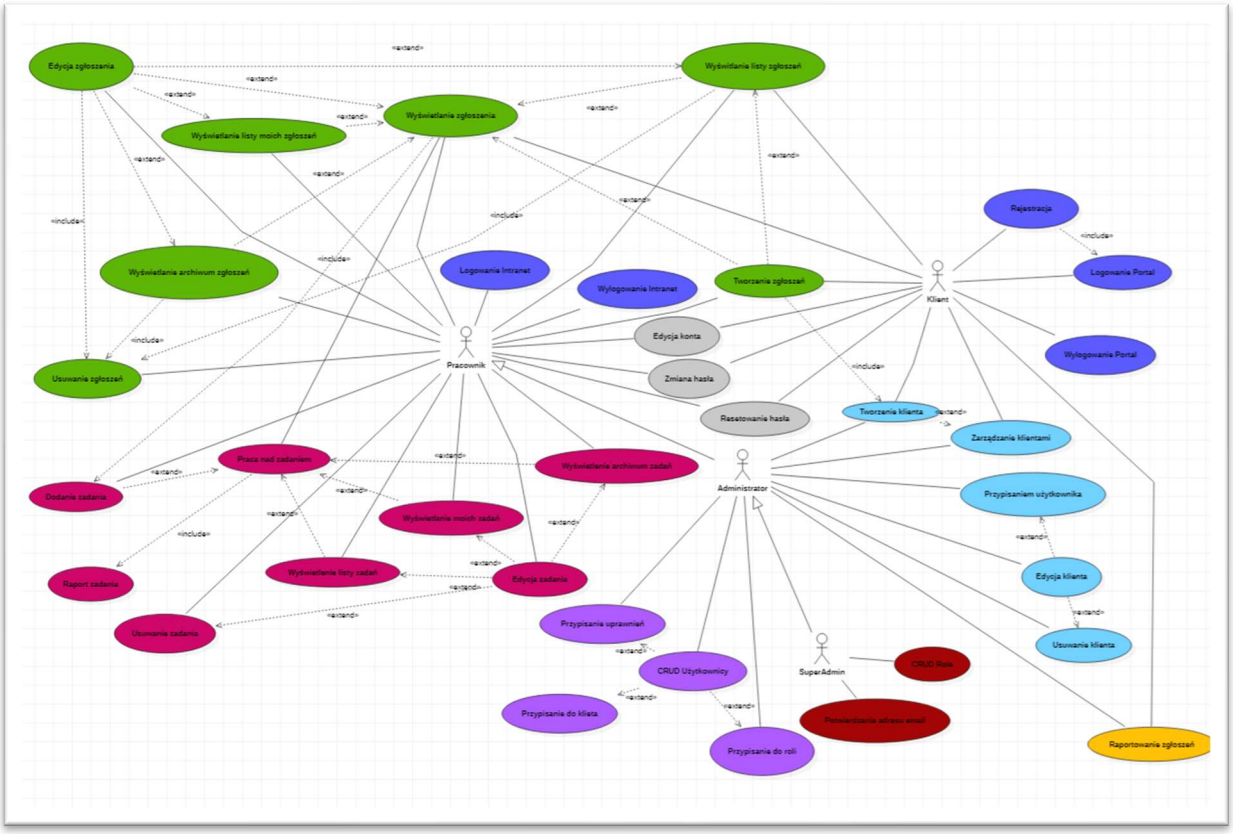
4.4 Blokada konta

Dodatkową funkcją zabezpieczającą konto przed atakiem typu „brute force” i wykradzeniem danych jest ustawienie blokady konta. Każdorazowe 5 prób błędnie wpisanego hasła do logowania kończy się blokadą konta na 15 minut, po tym czasie użytkownik ponownie posiada kolejne 5 prób logowania. Między czasie istnieje możliwość standardowego zresetowania hasła co pozwala na uzyskanie natychmiastowego dostępu.

5. Zakres funkcjonalny

5.1 Diagram przypadków użycia

Rysunek 11. Diagram przypadków użycia



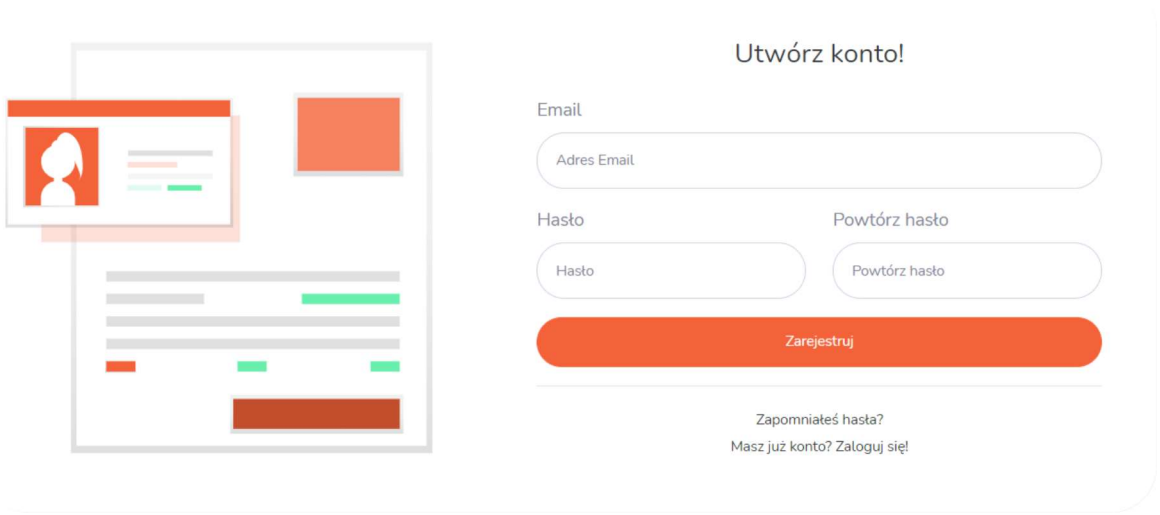
5.2 Przegląd funkcjonalności

Film prezentujący krótką funkcjonalność można zobaczyć w serwisie YouTube pod adresem: https://youtu.be/0ZC2msD8AjU

5.2.1 Pierwsza rejestracja i logowanie

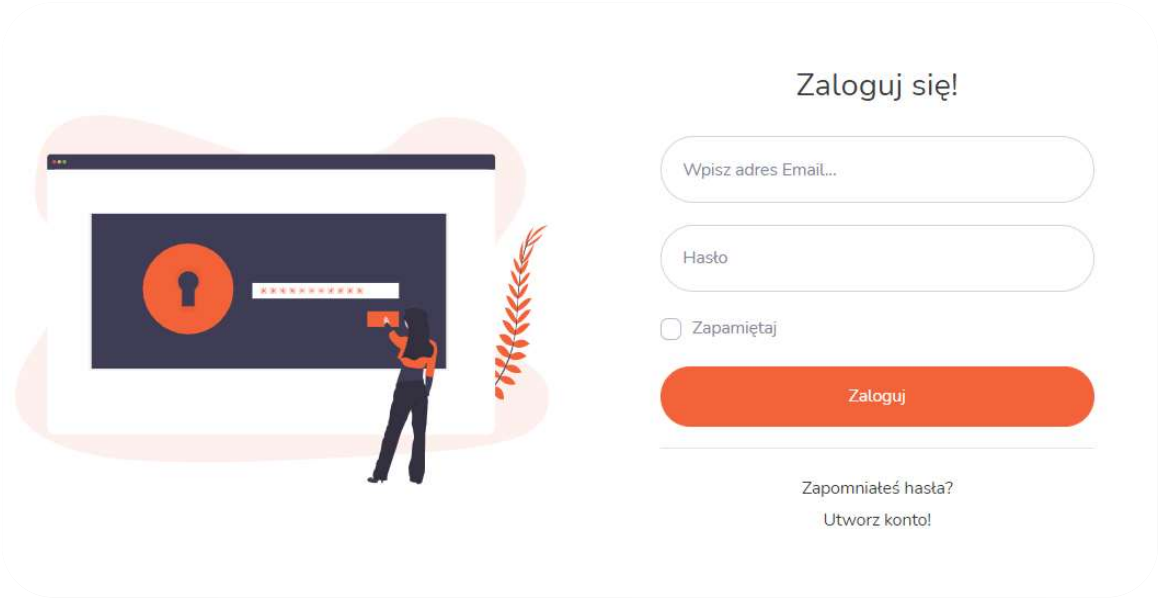
Dostęp do usług dostarczanych przez serwis wymaga rejestracji konta w systemie. Do portalu klienta może zalogować się każdy kto ukończył pomyślnie proces rejestracji. Konto dla pracownika można założyć jedynie z poziomu panelu administracyjnego.

Rysunek 12. Formularz utworzenia konta klienta



W obu przypadkach uzyskanie dostępu do systemu wymaga zalogowania się poprawnymi danymi.

Rysunek 13. Okno logowania

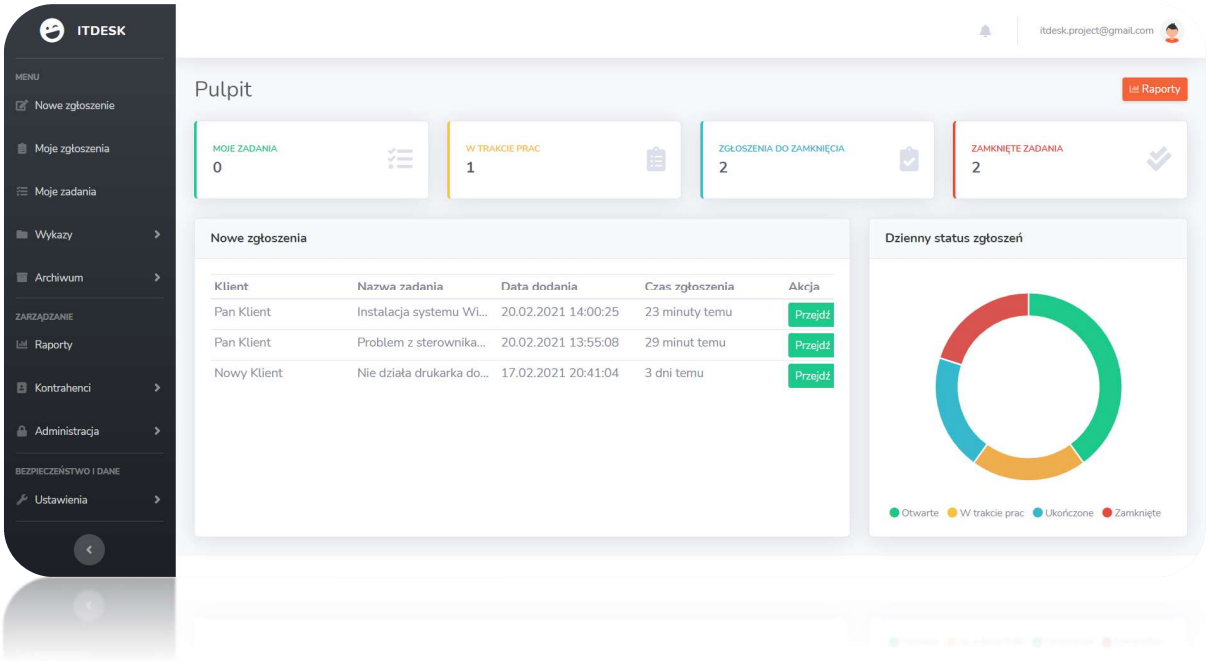


5.2.2 Widok główny panelu po stronie pracownika i klienta

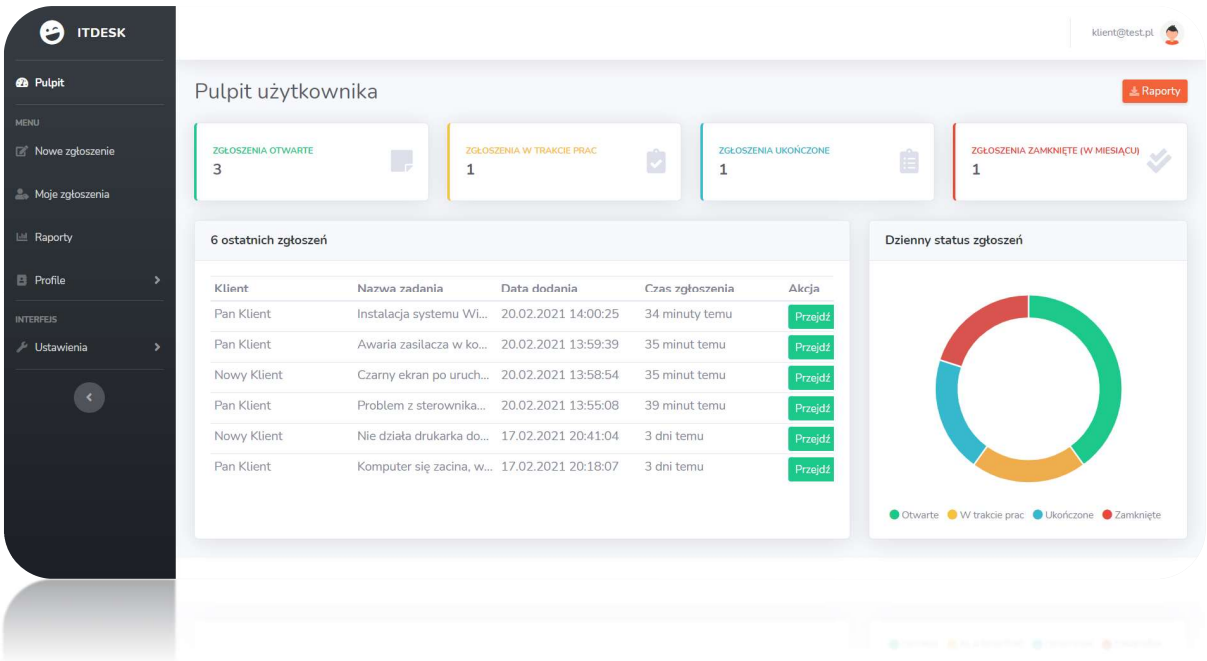
Strona główna dostępna zaraz po zalogowaniu umożliwia wgląd w podstawowe informacje dotyczące swoich zgłoszeń. W przypadku klienta i pracownika sprawdzane jest jaki użytkownik

jest zalogowany i wyświetlane są spersonalizowane dane przeznaczone wyłącznie dla danego użytkownika. Np. widżet (ang. widget) „Dzienny status zgłoszeń" widoczny dla pracownika prezentuje informacje o statusie wszystkich zgłoszeń dla wszystkich klientów danego dnia, natomiast klient widzi tylko liczbę i status zgłoszeń przypisanych do niego profili.

Rysunek 14. Pulpit główny dla pracownika



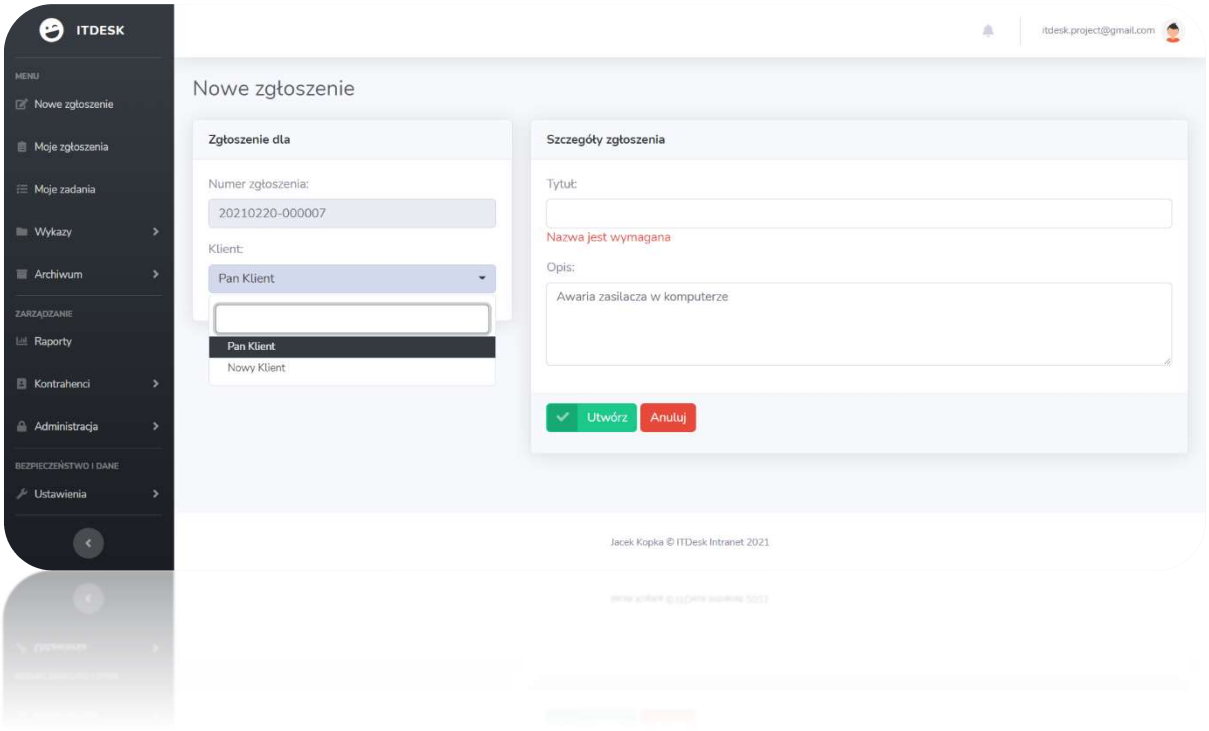
Rysunek 15. Pulpit główny dla klienta



5.2.3 Tworzenie zgłoszenia

Po konfiguracji odpowiednich uprawnień użytkownik uzyskuje podstawową funkcję systemu, czyli tworzenie zgłoszeń. W obu przypadkach okno „Nowe zgłoszenie" wygląda tak samo i daje dostęp do podstawowych informacji wymaganych przy tworzeniu zgłoszenia.

Rysunek 16. Tworzenie nowego zgłoszenia



Numer zgłoszenia nadawany jest automatycznie i składa się z aktualnej daty oraz wewnętrznego numeru nadawanego przez system. Funkcja sprawdzająca szuka ostatniego numeru zgłoszenia w bazie i nadaje kolejny dostępny numer zwiększając licznik o jeden. Dodanie zgłoszenia bez uzupełnienia podstawowych danych tj. tytuł, opis, klient jest niemożliwe zgłaszając to użytkownikowi stosownym komunikatem.

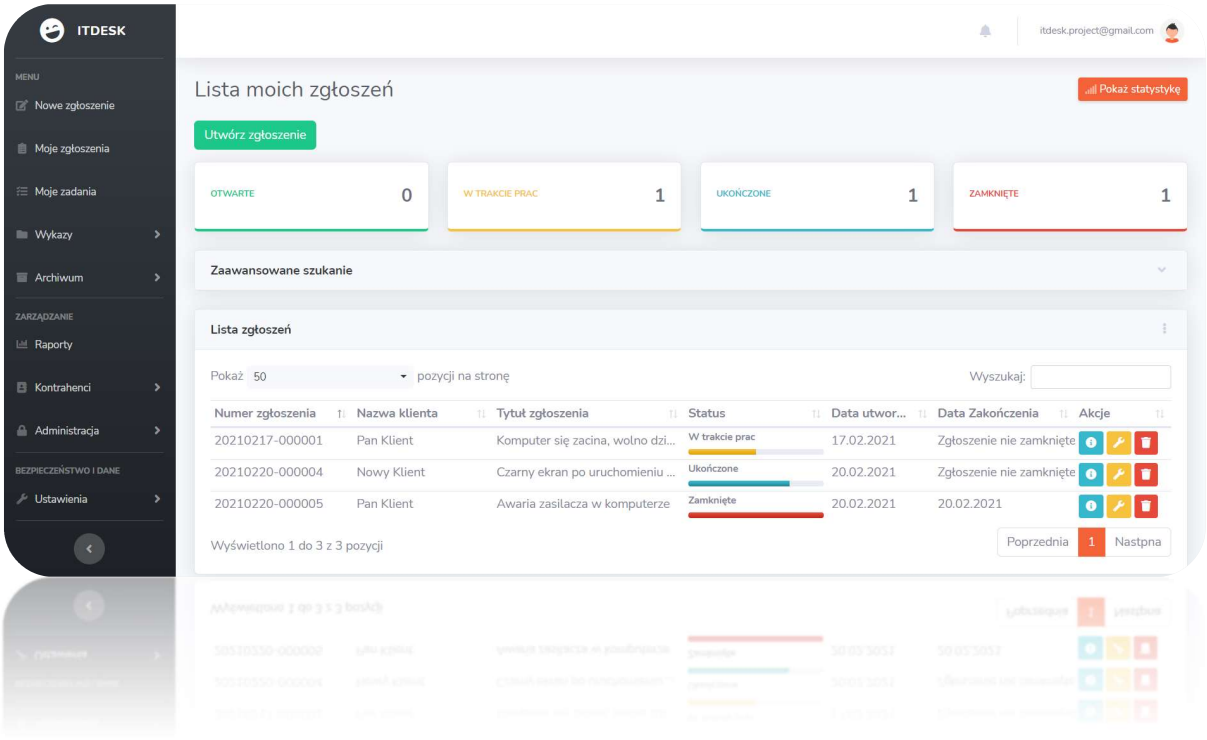
5.2.4 Obsługa zgłoszeń

Do dyspozycji pracownika zostały udostępnione predefiniowane widoki z gotowym zestawem przefiltrowanych danych. Mamy do dyspozycji widok:

- „Moje zgłoszenia" – dostęp do wszystkich zgłoszeń, gdzie w zadaniu przypisany jest zalogowany użytkownik,
- „Wykazy" -> „Zgłoszenia" – lista wszystkich niezamkniętych zgłoszeń, każdego użytkownika w systemie,

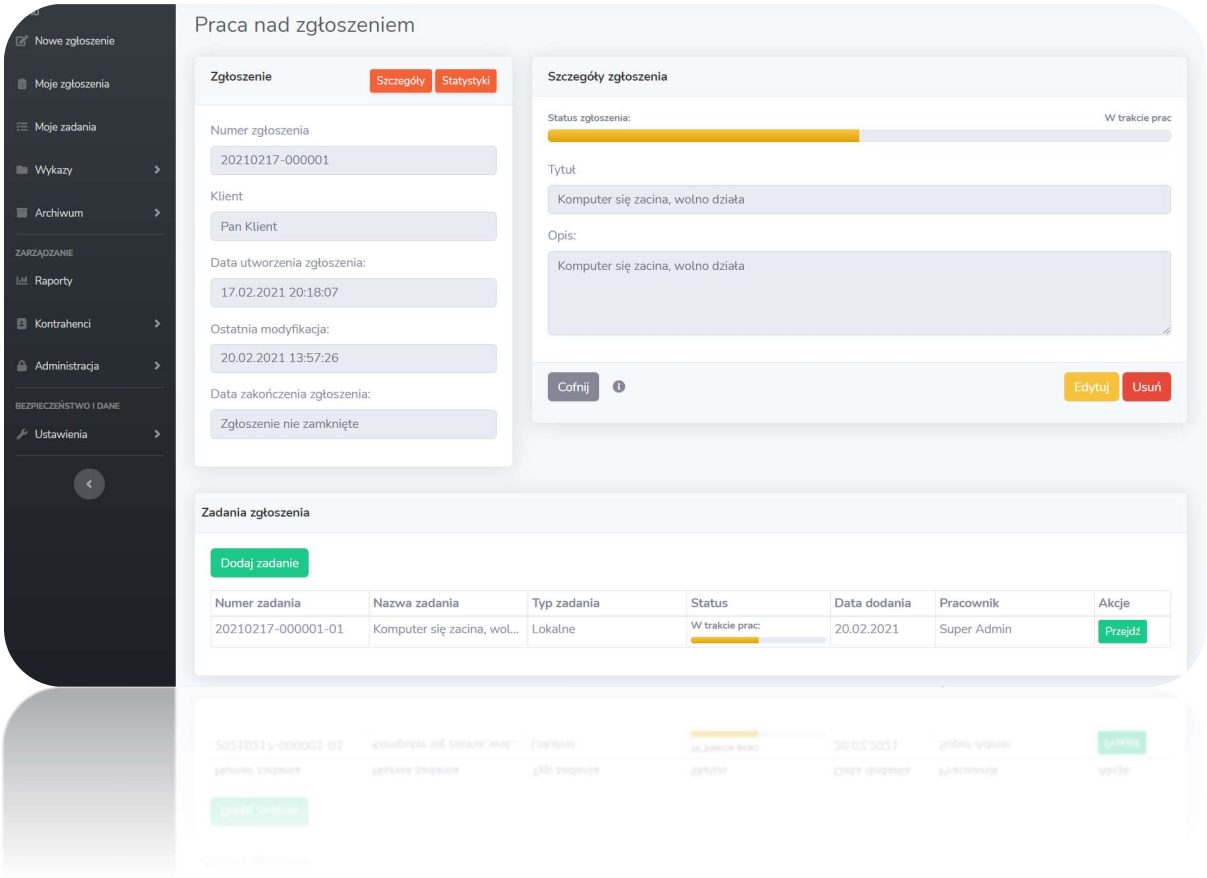
- „Archiwum” -> „Zgłoszenia” – wszystkie zgłoszenia, które posiadają status „Zamknięte”

Rysunek 17. Widok „Moje zgłoszenia”

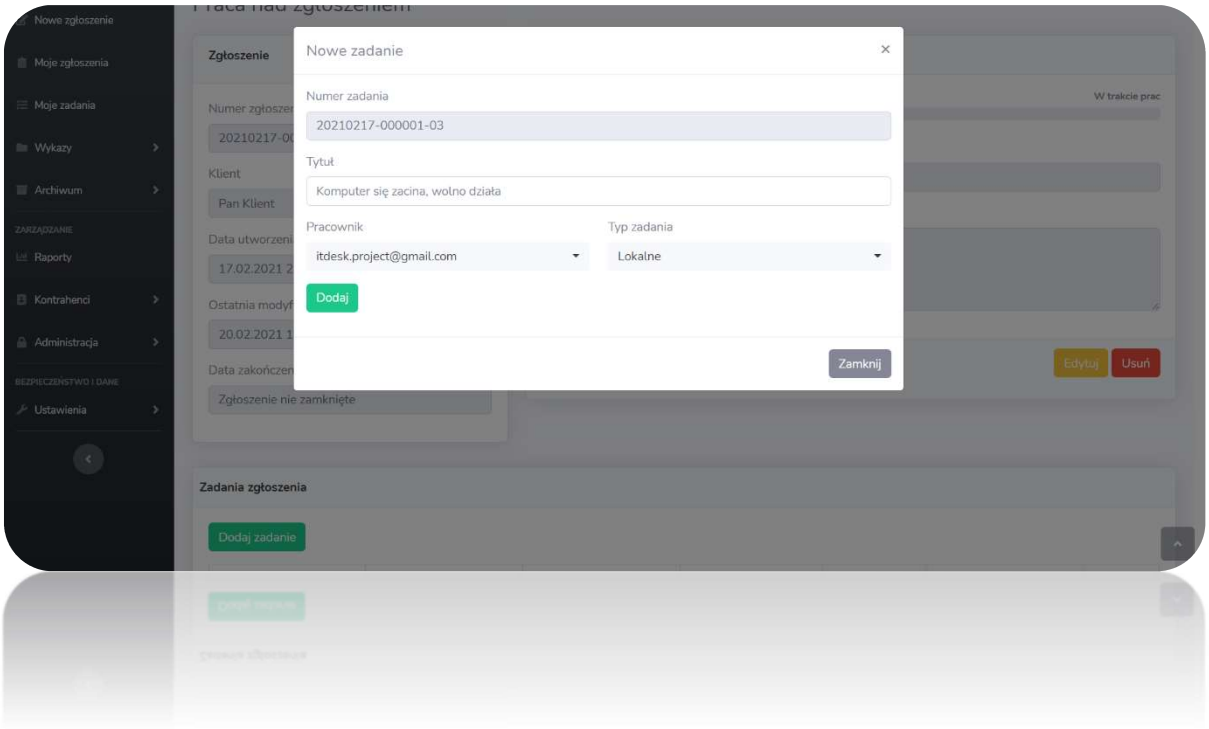


Otwarcie zgłoszenia powoduje przejście do widoku pracy nad zgłoszeniem. Pracownik uzyskuje informacje o podstawowych danych, możliwość edycji i usunięcia, zakończenia lub zamknięcia zgłoszenia. Jeśli status zgłoszenia jest inny niż „Zamknięte” możemy przypisać zadanie w zgłoszeniu wybierając pracownika oraz typ zadania. Po zapisie wybrany pracownik dostanie informację w systemie powiadomień i zgłoszenie trafi na jego listę zgłoszeń do obsługi. Ukończenie wszystkich zadań przypisanych w zgłoszeniu umożliwia zamknięcie zgłoszenia, status zgłoszenia zmieni się na „Zamknięte”, a klient dla którego wykonywane było zgłoszenie zostanie powiadomiony o tym fakcie (w zależności od aktywnej opcji) wiadomością e-mail oraz na swojej liście zgłoszeń w portalu klienta. Zamknięcie zgłoszenia, w którym jakieś zadanie nie zostało ukończone jest niemożliwe. Po zamknięciu zgłoszenia nie ma możliwości dodania nowych zadań oraz edycji zgłoszenia w takim wypadku należy ponownie otworzyć zgłoszenie.

Rysunek 18. Widok "Praca nad zgłoszeniem"



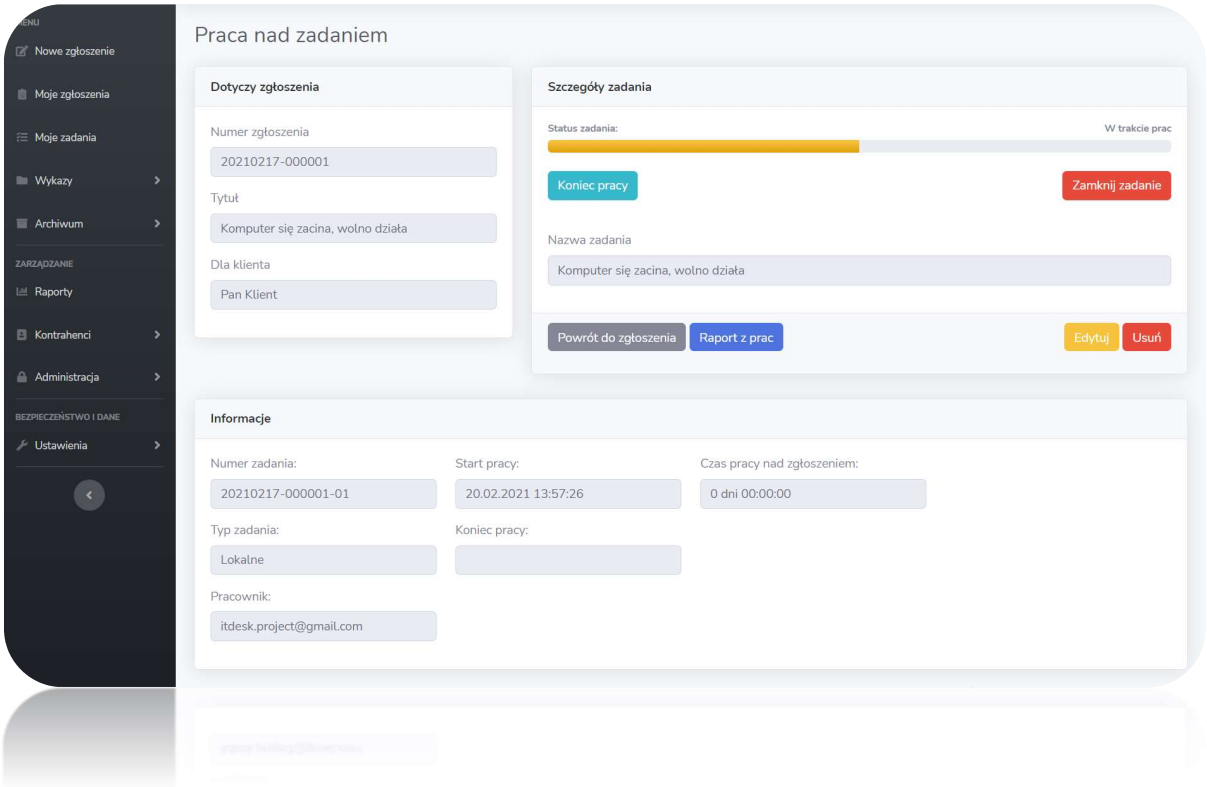
Rysunek 19. Dodawanie nowego zadania w zgłoszeniu



5.2.5 Praca nad zadaniem

Analogicznie jak w przypadku zgłoszenia widok „Praca nad zadaniem” dostarcza podstawowych informacji o zadaniu w zgłoszeniu. Status zadania zmienia się w momencie wywołania funkcji dla odpowiedniego typu zadania. System daje możliwość wyboru dwóch typów zadań: lokalnego i wyjazdowego. W przypadku pierwszego mamy czas liczony od startu pracy do jej zakończenia. Zadanie wyjazdowe odblokowuje dodatkowo licznik czasu dla dojazdu do zadania. Sumaryczny czas zadań uwzględniony jest na widoku zgłoszenia w sekcji „Statystyki”. Aby w przyszłości mieć podgląd w historię i to co zostało zrobione, pracownik może uzupełnić raport z zadania wpisując stosowne notatki.

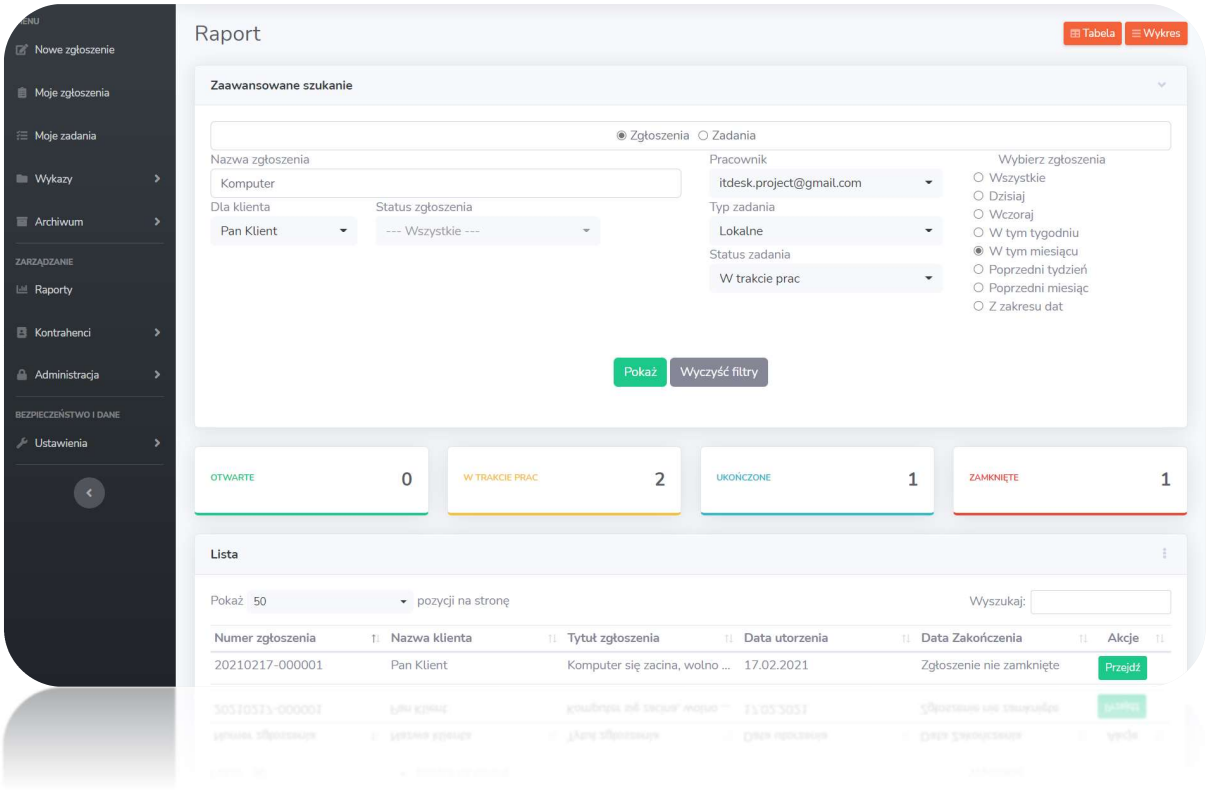
Rysunek 20. Widok "Praca nad zadaniem"



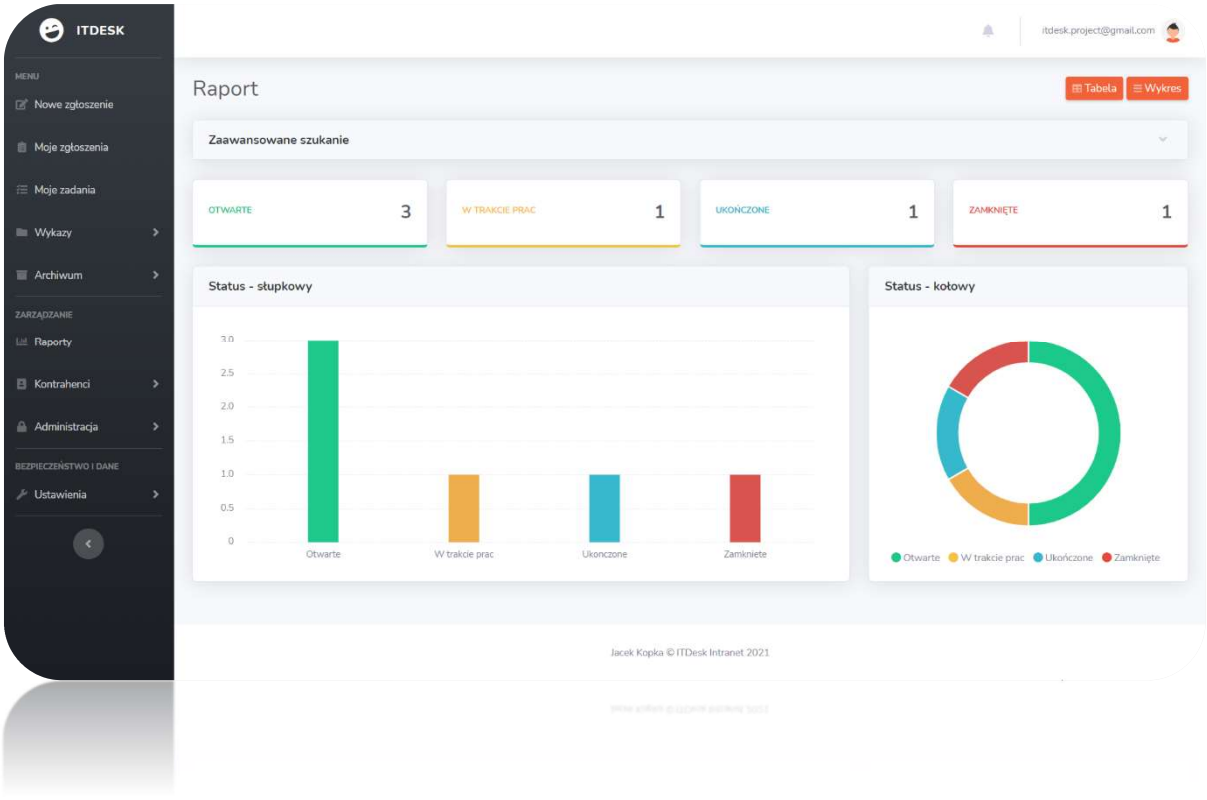
5.2.6 Raporty

Dla administracji systemu został przygotowany specjalny moduł raportów. Przedstawione wyniki można analizować w dowolny sposób w formie tabeli lub wykresów graficznych. Zastosowane filtry umożliwiają wybór zgłoszeń lub zadań, odpowiedniego klienta, pracownika, typ i status zadania, zgłoszenia oraz zakres dat w których wykonywane były zgłoszenia.

Rysunek 21. Sekcja raportów



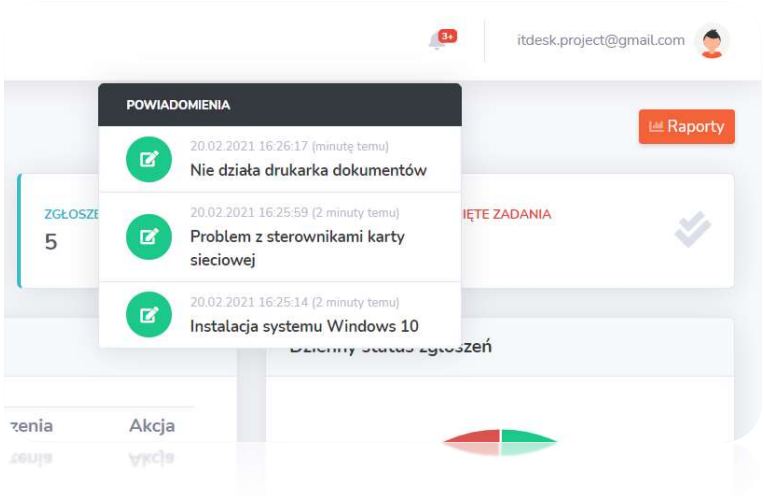
Rysunek 22. Graficzny status raportu



5.2.7 Powiadomienia

Powiadomienia email są zastosowane wyłącznie dla zmian dotyczących konta i statusu zgłoszeń dla klienta. Dodatkowym wewnętrznym mechanizmem są alerty dla pracowników. W górnej sekcji zaraz obok nazwy użytkownika widnieje ikona „dzwoneczka" informująca o przypisaniu nowego zgłoszenia. Kliknięcie w wybrany komunikat przekierowuje użytkownika do zgłoszenia, w którym przypisano to zadanie.

Rysunek 23. Wewnętrzne powiadomienia o nowym zadaniu

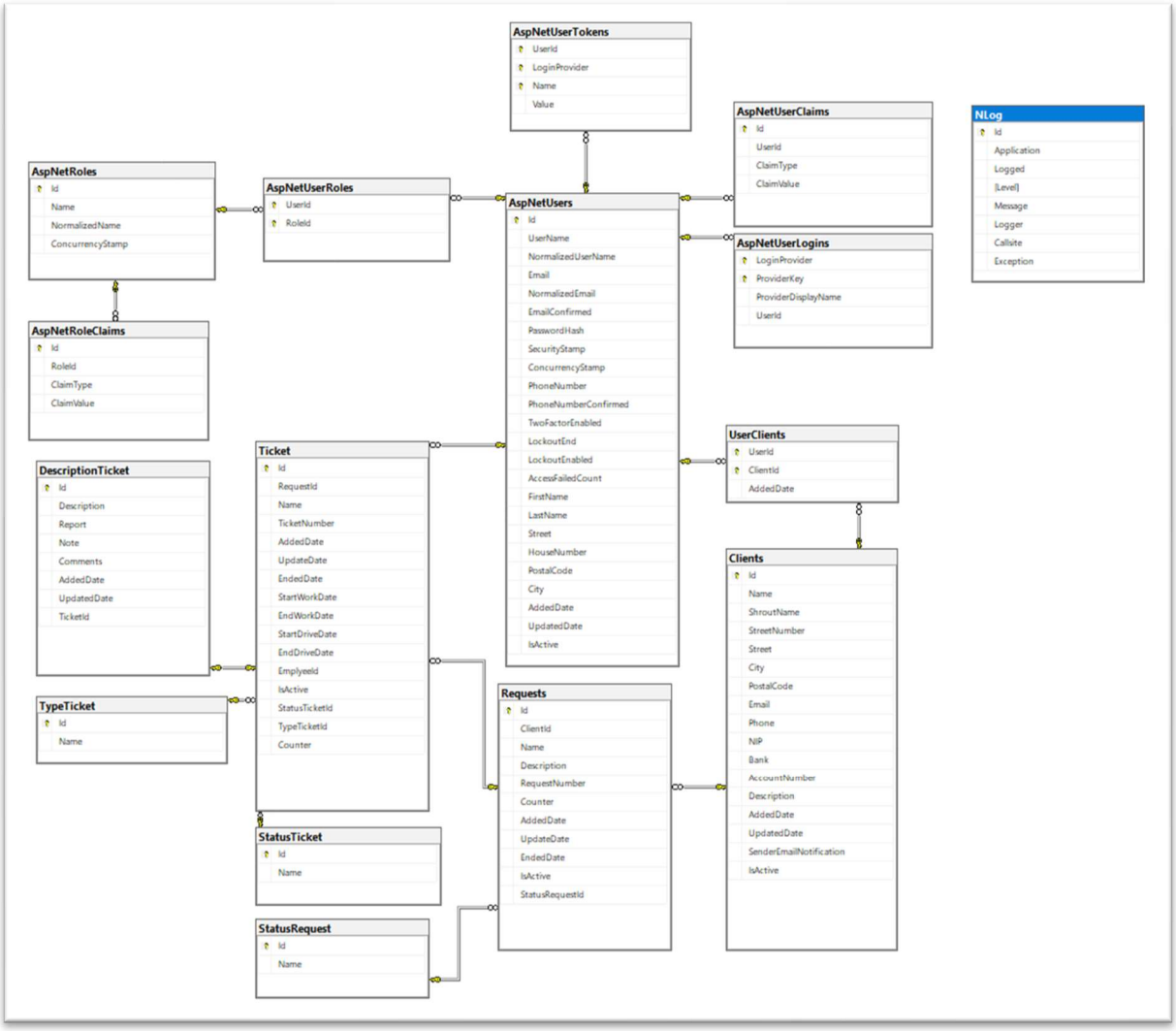


6. Wybrane szczegóły implementacyjne

6.1. Struktura bazy danych

Na strukturę bazy danych składa się odpowiednio 16 tabel. W tym jedna o nazwie „NLog" niepowiązana żadną relacją z pozostałymi, przechowująca zapis logów z aplikacji i monitoring aktywności użytkownika. Ponad to możemy wydzielić 7 tabel odpowiedzialnych za dane użytkowników aplikacji, 6 dotyczących zgłoszeń i zadań oraz 2 przechowujące informacje o profilach/kontrahentach. Poniższy rysunek prezentuje bazę danych wraz z połączeniami. Szczegółowe relacje zostały opisane w następnym rozdziale.

Rysunek 24. Struktura bazy danych



6.1.1 Relacje w bazie danych

Aby zapewnić integralność danych, sposób ich przekazywania oraz odczytu w bazie danych istnieją pewne powiązania między tabelami. Zbiór ich typu i zachodzącej relacji przedstawia się poniżej.

Legenda:

- Tabele – nazwa występującej tabeli,
- Relacje – liczba zachodzących relacji dla tabeli,
- Klucze_obce – liczba kluczy obcych w tabeli,
- Referencje – liczba referencji klucza obcego z innych tabel
- Powiązane_tabele – liczba różnych powiązanych tabel (niezależnie od typu relacji),
- Powiązania_tabeli - liczba tabel do których odwołuje się klucz obcy,
- Odniesienia_tabeli – liczba innych tabel odwołujących się do tej tabeli za pomocą kluczy obcych

Rysunek 25. Spis zachodzących relacji w bazie danych – opracowanie własne

	Tabele	Relacje	Klucze_obce	Referencje	Powiązane_tabele	Powiązania_tabeli	Odniesienia_tabeli
1	dbo.AspNetUsers	6	0	6	6	0	6
2	dbo.Ticket	5	4	1	5	4	1
3	dbo.Requests	3	2	1	3	2	1
4	dbo.AspNetUserRoles	2	2	0	2	2	0
5	dbo.UserClients	2	2	0	2	2	0
6	dbo.Clients	2	0	2	2	0	2
7	dbo.AspNetRoles	2	0	2	2	0	2
8	dbo.AspNetUserClaims	1	1	0	1	1	0
9	dbo.AspNetUserLogins	1	1	0	1	1	0
10	dbo.AspNetRoleClaims	1	1	0	1	1	0
11	dbo.DescriptionTicket	1	1	0	1	1	0
12	dbo.AspNetUserTokens	1	1	0	1	1	0
13	dbo.TypeTicket	1	0	1	1	0	1
14	dbo.StatusRequest	1	0	1	1	0	1
15	dbo.StatusTicket	1	0	1	1	0	1

Użytkownik w systemie jest przypisany do roli, zachodzi zależność pomiędzy tabelami **AspNetUsers** i **AspNetRoles** łącząc klucze główne w tabeli **AspNetUsersRoles**. Chodź istnieje w tym wypadku relacja „wiele do wielu” możliwość przypisania wielu ról ze względów technicznych została zablokowana. Ponadto każdy użytkownik ma przypisany zbiór uprawnień relacją „wiele do wielu”. Podobne powiązanie występuje w przypadku użytkownik <-> klient

– każdy użytkownik może posiadać wielu klientów i każdy klient może być przypisany do wielu użytkowników.

6.2 Nowe zgłoszenie

Metoda HttpGet „Create” wywoływana z akcji kontrolera „RequestController”, zwraca widok utworzenia nowego zgłoszenia wraz z przeniesieniem kluczowych parametrów poprzez mechanizm ViewData – możliwość przekazywania informacji na krótki czas. W pierwszej kolejności jest pobierany numer zgłoszenia wywołując funkcję „RequestNumber()”, która pobiera z bazy danych ostatni numer zgłoszenia, zwiększa licznik o jeden i nadaje aktualnie tworzonemu zgłoszeniu. W wartości ViewData[„ClientId”] przekazywana jest lista aktywnych klientów identyfikując rekordy po kolumnie Id.

Wywołanie Metoda HttpPost „Create” przyciskiem „Zapisz” dodaje do bazy danych nowe zgłoszenie o parametrach zwróconych metodą Get. Na początku sprawdzany jest ponownie numer zgłoszenia funkcją „RequestNumber()” przed dokonaniem zapisu i porównywany z aktualnym numerem na widoku. Mechanizm ten zabezpiecza dodanie zgłoszeń o tym samym numerze. Utworzenie nowego zgłoszenia automatycznie nadaje mu status „1” – „Otwarte”, wartość IsActive jest ustawiana na „true” oraz pola AddedDate i UpdatedDate mają przypisaną aktualną datę i godzinę wywołując właściwość „DateTime.Now”. Dodatkowo jeżeli na kliencie dla którego tworzone jest zgłoszenie wartość pola „SenderEmailNotification” jest „true” wysyłana jest wiadomość email na adres klienta zaimplementowaną wcześniej metodą „SmtpConf”.

Po pomyślnym rezultacie wykonania metody dodatkowo jest tworzony wpis do tabeli „NLog” z informacją o utworzeniu zgłoszenia z numerem zgłoszenia i Id osoby tworzącej.

Przed deklaracją metody jest atrybut [Authorize(Policy = „CreateRequestPolicy”)] związany z opisanym powyżej interfejsem ASP.NET Identity, który sprawdza uprawnienia dostępu do akcji kontrolera dla użytkownika, który próbuje go wywołać.

Rysunek 26. Metoda Create kontrolera RequestController

```
[HttpPost]
[ValidateAntiForgeryToken]
[Authorize(Policy = "CreateRequestPolicy")]
public async Task<IActionResult>
Create([Bind("Id,RequestNumber,Name,Description,AddedDate,UpdateDate,EndedDate,IsActive,StatusRequestId,ClientId")] Request request)
{
    var requestNumber = RequestNumber();

    var loginUser = await userManager.GetUserAsync(User);
    var client = await _context.Clients.FirstOrDefaultAsync(c => c.Id == request.ClientId);

    if (ModelState.IsValid)
    {
        if (request.RequestNumber != requestNumber)
        {
            request.RequestNumber = requestNumber;
        }
        request.StatusRequestId = "1";
        request.IsActive = true;
        request.AddedDate = DateTime.Now;
        request.UpdateDate = DateTime.Now;
        _context.Add(request);
        await _context.SaveChangesAsync();

        if (client.SenderEmailNotification)
        {
            SmtpConf.AddressTo = client.Email;
            SmtpConf.MessageSubject = "ITDesk - Nowe zgłoszenie";
            SmtpConf.MessageContent = "Pojawiło się nowe zgłoszenie zarejestrowane na klienta  " +  client.ShroutName + " o numerze "
                                    + request.RequestNumber + "<br>"
                + "Zaloguj się na swoje konto w Portalu WWW, żeby móc je śledzić."
                + "<br>" + "<br>"
                + "Pozdrawiamy <br>"
                + "Zespół ITDesk";

            SmtpConf.sendMail();
        }

        logger.Log(LogLevel.Information, "Utworzono zgłoszenie " + request.RequestNumber + " przez " + loginUser.Id
                  + ", Wysyłanie maila:" + client.SenderEmailNotification + ", Status wysłania: " + SmtpConf.SendedEmail);

        return RedirectToAction(nameof(Index));
    }

    logger.Log(LogLevel.Warning, "Błąd w tworzeniu zgłoszenia");

    ViewData["ClientId"] = new SelectList(_context.Clients, "Id", "ShroutName", request.ClientId);
    ViewData["StatusRequestId"] = new SelectList(_context.StatusRequest, "Id", "Name", request.StatusRequestId);
    return View(request);
}
```

6.3 Logowanie

Podstawowa funkcja uwierzytelniająca użytkownika w systemie jako parametry przyjmuje dane z modelu, czyli adres email i hasło oraz adres URL strony. Pierwszą pozycją do sprawdzenia jest to czy użytkownik który próbuje się zalogować ma potwierdzony adres email w systemie, w takim wypadku zostanie zwrócony komunikat o nieaktywowaniu konta. Następnie jest zablokowana możliwość logowania użytkowników, którzy mają przypisaną rolę „Klient” wyklucza to możliwość zalogowania się klienta do portalu pracownika.

Za sprawdzenie poprawności zalogowania odpowiada wbudowana funkcja interfejsu ASP.NET Identity PasswordSignInAsync().

Rysunek 27. Funkcja PasswordSignInAsync()

```
var result = await signInManager.PasswordSignInAsync(model.Email, model.Password, model.RememberMe, true);
```

Zwrócenie wyniku „Suceedded" wykonania tej metody przekierowuje użytkownika bezpośrednio do strony głównej aplikacji zapisując dodatkowo informację w bazie o pomyślnym logowaniu. W przypadku wartości „IsLockedOut" – konto użytkownika jest zablokowane wyświetla się strona z informacją o blokadzie konta.

Każde nieudane logowanie zapisuje do bazy informację o próbie wykonania i przyczynę z jakieś nie uzyskano dostępu do systemu.

Rysunek 28. Metoda Login kontrolera AccountController

```
[HttpPost]
[AllowAnonymous]
public async Task<IActionResult> Login(Login model, string returnUrl)
{
    if (ModelState.IsValid)
    {
        var user = await userManager.FindByEmailAsync(model.Email);

        if (user != null && !user.EmailConfirmed && (await userManager.CheckPasswordAsync(user, model.Password)))
        {
            ModelState.AddModelError(string.Empty, "Potwierdź adres Email");
            logger.Log(LogLevel.Warning, "Użytkownik " + user.Id + " nie ma potwierdzonego maila");
            return View(model);
        }

        if(user != null && await userManager.IsInRoleAsync(user, "Klient"))
        {
            ModelState.AddModelError(string.Empty, "Nieprawidłowa próba logowania");
            logger.Log(LogLevel.Warning, "Nieprawidłowa próba logowania użytkownika " + user.Id);
            return View(model);
        }

        var result = await signInManager.PasswordSignInAsync(model.Email, model.Password, model.RememberMe, true);

        if (result.Succeeded && user.IsActive)
        {
            if (!string.IsNullOrEmpty(returnUrl) && Url.IsLocalUrl(returnUrl))
            {
                return Redirect(returnUrl);
            }
            else
            {
                logger.Log(LogLevel.Warning, "Pomyślne logowanie użytkownika " + user.Id);
                return RedirectToAction("index", "home");
            }
        }

        if (result.IsLockedOut)
        {
            logger.Log(LogLevel.Warning, "Zablokowano konto użytkownika " + user.Id);
            return View("AccountLocked");
        }
        ModelState.AddModelError(string.Empty, "Nieprawidłowa próba logowania");
        if (user == null)
        {
            logger.Log(LogLevel.Warning, "Nieprawidłowa próba logowania użytkownika " + model.Email
                      + " |Przyczyna: brak użytkownika w systemie");
        }
        else
        {
            logger.Log(LogLevel.Warning, "Nieprawidłowa próba logowania użytkownika " + model.Email);
        }
    }

    return View(model);
}
```

6.4 Resetowanie hasła

Inicjacją funkcji resetowania hasła jest wcześniejsze wywołanie akcji „ForgotPassword()” w której na podany adres email wysyłany jest specjalny link resetujący na który składa się przekierowanie na stronę ../Account/ReserPassword. W linku zaszyty jest adres email osoby

dla której wykonywana jest operacja i specjalny wygenerowany token zabezpieczający (czas życia tokenu wynosi 6 godzin).

Analizując kod źródłowy akcji odpowiedzialnej za resetowanie hasła widzimy, że jako parametr przyjmuje „model" typu „ResetPassword" w którym mamy dostępne 4 właściwości string Email, string Password, string ConfirmPassword oraz string Token. W pierwszej kolejności pobierany jest obiekt typu „ApplicationUser" – użytkownik dla którego ma zostać zmienione hasło identyfikując go po adresie email. Następnie po spełnieniu warunków, że użytkownik istnieje i wartość tokenu nie jest pusta wywoływana jest funkcja „ResetPasswordAsync()". Pomyślny rezultat jej wykonania zeruje blokadę na koncie użytkownika (jak wspomniano w rozdziale **4.4 Blokada konta**, po zablokowaniu konta przez 5 nieudanych prób logowania, użytkownik musi odczekać 15 minut lub skorzystać z funkcji resetowania hasła) i wysyła informację na skrzynkę pocztową o zmianie hasła. Odpowiednia informacja o wykonanej aktywności jest zapisywana w bazie danych. Niepowodzenie wykonania metody zgłaszane jest dla użytkownika odpowiednim komunikatem na ekranie.

Rysunek 29. Metoda ResetPassword kontrolera AccountController

```
[HttpPost]
[AllowAnonymous]
public async Task<IActionResult> ResetPassword(ResetPassword model)
{
    var user = await userManager.FindByEmailAsync(model.Email);

    if (ModelState.IsValid)
    {
        if (user == null)
        {
            ModelState.AddModelError(string.Empty, "Nieprawidłowy adres email");
            return View();
        }
        if (user != null && model.Token != null)
        {
            var result = await userManager.ResetPasswordAsync(user, model.Token, model.Password);
            if (result.Succeeded)
            {
                if(await userManager.IsLockedOutAsync(user))
                {
                    await userManager.SetLockoutEndDateAsync(user, DateTimeOffset.UtcNow);
                }

                logger.Log(LogLevel.Warning, "Reset hasła użytkownika " + user.Id);

                SmtpConf.AddressTo = user.Email;
                SmtpConf.MessageSubject = "ITDesk - Potwierdzenie zresetowania hasła";
                SmtpConf.MessageContent = "Twoje hasło zostało zresetowane. <br> "
                    + "<br>" + "<br>"
                    + "Pozdrawiamy <br>"
                    + "Zespół ITDesk";

                SmtpConf.sendMail();

                return View("ResetPasswordConfirmation");
            }
            foreach (var error in result.Errors)
            {
                ModelState.AddModelError("", error.Description);
                logger.Log(LogLevel.Warning, "Nie udało się zresetować hasła użytkownika "
                          + user.Id + " " + error.Description);
            }
            return View(model);
        }
        ModelState.AddModelError(string.Empty, "Sprawdź link");
        return View("ResetPasswordConfirmation");
    }

    logger.Log(LogLevel.Warning, "Nie udało się zresetować hasła użytkownika " + user.Id);
    return View(model);
}
```

6.5 Autoryzacja użytkownika

Pomijając metodę logowania jako pierwszej identyfikacji użytkownika zalogowanego w systemie wprowadzenie podziału na role i uprawnienia pozwoliło na kontrolowanie dostępu do wybranych funkcji aplikacji w zależności od potrzeb i stanowiska pracownika w firmie. Wybranie odpowiedniej przynależności definiuje to co użytkownik może, a czego nie może zrobić.

Zdefiniowanie atrybutu [Authorize] w każdym kontrolerze wymusza autoryzację osoby, która chce uzyskać dostęp do akcji kontrolera, ponadto wszystkie akcje posiadają dodatkowo atrybut sprawdzający uprawnienie dostępu do danej funkcji. Przeanalizujmy poniższe przykłady

Rysunek 30. Fragment kodu kontrolera AdministrationController

```
[Authorize]
public class AdministrationController : Controller
{
    private readonly RoleManager<IdentityRole> roleManager;
    private readonly UserManager<ApplicationUser> userManager;
    private readonly ILogger<AdministrationController> logger;
    private readonly ITDeskContext _context;

    SmtpConfiguration SmtpConf = new SmtpConfiguration();
    public AdministrationController(RoleManager<IdentityRole> roleManager,
                                    UserManager<ApplicationUser> userManager,
                                    ILogger<AdministrationController> logger,
                                    ITDeskContext context)...

    [HttpGet]
    [Authorize(Policy = "ManageClientInUser")]
    public async Task<IActionResult> ManageUserClient(string userId)...


    [HttpPost]
    [Authorize(Policy = "ManageClientInUser")]
    public async Task<IActionResult> ManageUserClient(List<Data.Helpers.UserClients> model, string userId)...

    [HttpGet]
    [AllowAnonymous]
    public IActionResult AccessDenied()...

    [HttpGet]
    [Authorize(Policy = "EditClaimsUserPolicy")]
    public async Task<IActionResult> ManageUserClaims(string userId)...

    [HttpPost]
    [Authorize(Policy = "EditClaimsUserPolicy")]
    public async Task<IActionResult> ManageUserClaims(UserClaims model)...

    ...
}
```

Rysunek 31. Fragment pliku Startup.cs

```
options.AddPolicy("ManageUserInClient", policy => policy.RequireAssertion(context =>
                  context.User.IsInRole("Administrator") &&
                  context.User.HasClaim(claim => claim.Type == "Zarządzanie użytkownikami na kliencie" && claim.Value == "true") ||
                  context.User.IsInRole("SuperAdmin")
              ));
```

W pliku „Startup.cs” dodanie opcji usługi „services.AddAuthorization()” o roboczej nazwie „ManageUserInClient” wymusza przypisanie użytkownika do roli „Administrator” i spełnienie warunku uprawnień „Zarządzanie uprawnieniami na użytkowniku” lub wyłącznie

przynależność do roli „SuperAdmin”. Rezultatem jest uzyskanie dostępu do funkcji „ManageUserClient” w klasie kontrolera „AdministrationController”.

Innym zastosowaniem autoryzacji jest odmowa dostępu do edycji własnych uprawnień przez zalogowanego użytkownika. Najprościej rzecz ujmując wyklucza to możliwość nadania lub odebrania samemu sobie uprawnień i przynależności do roli. Dotyczy to wyłącznie roli Administratora systemu z wykluczeniem roli SuperAdmin, inni użytkowni nie mają dostępu do wybranych funkcji.

Rysunek 32. Fragment pliku Startup.cs

```
...
options.AddPolicy("EditRoleUserPolicy", policy => policy.AddRequirements(new ManageAdminRolesAndClaimsRequirment()));
...
});

services.AddSingleton<IAuthorizationHandler, CanEditOnlyOtherAdminClaimsHandler>();
...
}
```

Rysunek 33. Funkcja CanEditOnlyOtherAdminClaimsHandler

```
public class CanEditOnlyOtherAdminClaimsHandler : AuthorizationHandler<ManageAdminRolesAndClaimsRequirment>
{
    private readonly IHttpContextAccessor httpContextAccessor;

    public CanEditOnlyOtherAdminClaimsHandler(IHttpContextAccessor httpContextAccessor)
    {
        this.httpContextAccessor = httpContextAccessor ?? throw new ArgumentNullException(nameof(httpContextAccessor));
    }

    protected override Task HandleRequirementAsync(AuthorizationHandlerContext context,
                                                   ManageAdminRolesAndClaimsRequirment requirement)
    {
        if (context.User == null || !context.User.Identity.IsAuthenticated)
        {
            context.Fail();
            return Task.CompletedTask;
        }

        string loggedInAdminId = context.User.Claims.FirstOrDefault(c => c.Type == ClaimTypes.NameIdentifier).Value;
        string adminInBegingEdited = httpContextAccessor.HttpContext.Request.Query["userId"].ToString();

        if (context.User.IsInRole("Administrator")
            && context.User.HasClaim(claim => claim.Type == "Zarządzanie uprawnieniami na użytkowniku" && claim.Value == "true") &&
            adminInBegingEdited.ToLower() != loggedInAdminId.ToLower())
        {
            context.Succeed(requirement);
        }

        return Task.CompletedTask;
    }
}
```

Ostatnim rodzajem atrybutu wykorzystanym w kodach źródłowych jest [AllowAnonymous], który pomija wszelkie polityki autoryzacji i pozwala uzyskać dostęp dla anonimowych użytkowników. Przykładem może być akcja „Login()" lub „AccessDenied()".

6.6 Przypisywanie do roli i uprawnień

Opisywane akcje są dostępne wyłącznie dla Administratorów systemu z odpowiednimi uprawnieniami lub roli SuperAdmin. Z założenia każdy użytkownik może być przypisany wyłącznie do jednego typu roli.

Ponownie z pomocą przychodzą wbudowane funkcje inerfejsu ASP.NET Identity. Obie akcje stosują podobne mechanizmy działania. Przeanalizujmy poniższy przykład przypisania użytkownika do roli

Rysunek 34. Metoda HttpGet ManageUserRoles kontrolera AdministrationController

```
[HttpGet]
[Authorize(Policy = "EditRoleUserPolicy")]
public async Task<IActionResult> ManageUserRoles(string userId)
{
    ViewBag.userId = userId;

    var user = await userManager.FindByIdAsync(userId);

    if (user == null)
    {
        ViewBag.ErrorMessage = $"Użytkownik o Id = {userId} nie został znaleziony";
        return View("NotFound");
    }

    var model = new List<UserRoles>().ToList();

    foreach (var role in roleManager.Roles)
    {
        var userRoles = new UserRoles
        {
            RoleId = role.Id,
            RoleName = role.Name
        };

        if (await userManager.IsInRoleAsync(user, role.Name))
        {
            userRoles.IsSelected = true;
        }
        else
        {
            userRoles.IsSelected = false;
        }

        model.Add(userRoles);
    }

    return View(model);
}
```

Jako parametr przyjmujemy Id użytkownika dla którego dokonywane są zmiany przypisania; w pierwszej kolejności sprawdzamy czy taki użytkownik istnieje i pobieramy listę dostępnych ról w systemie wraz z istniejącym przypisaniem danego użytkownika do roli. Rezultat jest zwracany i przekazywany do widoku.

Rysunek 35. Metoda HttpPost ManageUserRoles kontrolera AdministrationController

```
[HttpPost]
[Authorize(Policy = "EditRoleUserPolicy")]
public async Task<IActionResult> ManageUserRoles(List<UserRoles> model, string userId)
{
    var loginUser = await userManager.GetUserAsync(User);
    var user = await userManager.FindByIdAsync(userId);

    if (user == null)
    {
        ViewBag.ErrorMessage = $"Użytkownik o Id = {userId} nie został znaleziony";
        return View("NotFound");
    }

    var roles = await userManager.GetRolesAsync(user);
    var result = await userManager.RemoveFromRolesAsync(user, roles);

    logger.Log(LogLevel.Warning, "Usunięto role dla użytkownika " + user.Id + " przez " + loginUser.Id);

    if (!result.Succeeded)
    {
        ModelState.AddModelError("", "Nie można usunąć istniejących ról użytkownika");
        return View(model);
    }

    result = await userManager.AddToRolesAsync(user, model.Where(x => x.IsSelected).Select(y => y.RoleName));
    logger.Log(LogLevel.Warning, "Dodano role dla użytkownika " + user.Id + " przez " + loginUser.Id);

    if (!result.Succeeded)
    {
        ModelState.AddModelError("", "Nie można dodać wybranych ról do użytkownika");
        return View(model);
    }

    return RedirectToAction("EditUser", new { Id = userId });
}
```

Zmiany na widoku i zapis ustawień wywołuję akcję HttpPost, która pobiera aktualne role użytkownika i usuwa przypisanie.

Rysunek 36. Fragment kodu akcji HttpPost ManageUserRoles

```
...
var roles = await userManager.GetRolesAsync(user);
var result = await userManager.RemoveFromRolesAsync(user, roles);
...
```

I dodaje nowo przypisaną rolę wybraną na widoku do użytkownika.

Rysunek 37. Fragment kodu akcji HttpPost ManageUserRoles

```
...
result = await userManager.AddToRolesAsync(user, model.Where(x => x.IsSelected).Select(y => y.RoleName));
...
```

Wynik warunku „result.Succeeded” przyjmując wartość „false” oznacza błąd w wykonaniu metody i zwraca odpowiedni komunikat dla użytkownika. Informacja o zmianach na danym użytkowniku zostaje zapisana w bazie danych.

7. Pozostałe rozwiązania

7.1 Logowanie aktywności w aplikacji

Dokonując jakichkolwiek zmian w systemie, dodawania nowego użytkownika, zgłoszenia, zadania, edycja danych jest monitorowana i zapisywana w bazie z odpowiednią adnotacją. Uruchomienie aplikacji lokalnie na komputerze utworzy dodatkowo folder w systemie na dysku C:\ o nazwie „DemoLogs" z plikami logowania aktywności użytkownika.

Informacje zawarte w logach pozwalają przeanalizować sytuację i kroki jakie użytkownik aplikacji wykonał w sytuacjach awaryjnych.

Rysunek 38. Fragment metody zapisującej informacje o aktywności w bazie danych

```
...
logger.Log(LogLevel.Information, "Utworzono zgłoszenie " + request.RequestNumber + " przez " + loginUser.Id + ",
   Wysyłanie maila: " +  client.SenderEmailNotification + ", Status wysłania: " + SmtpConf.SendedEmail);
...
```

W strukturze pliku znajduje się aktualna data i czas zgłoszenia, z jakiej aplikacji wywoływana była funkcja, waga informacji, sama informacja – to co zostało wykonane oraz jaki kontroler był wywołany do wykonania akcji.

7.2 Testowanie i wdrożenie na platformie Azure

Nieodłącznym elementem tworzenia aplikacji było systematyczne testowanie dodanych funkcjonalności i wprowadzonych zmian, aby zapewnić najlepszą jakość dla klienta końcowego.

Dla przeprowadzenia dokładnych analiz podzielono aplikację na kilka odrębnych modułów przeprowadzając wewnętrzne symulacje wykonywania kodu przewidując zachowania użytkownika podczas pracy systemu w środowisku produkcyjnym. W pewnych wypadkach celowo „wprowadzano błędne dane", aby sprawdzić zabezpieczenia w kodzie. Testowanie aplikacji na mniejszych fragmentach pozwoliło na szybkie diagnozowanie i naprawę błędów. Nieocenioną pomocą okazało się zaimplementowanie we wczesnej fazie tworzenia aplikacji systemu logowania aktywności (patrz rozdział **7.1 Logowanie aktywności w aplikacji**), które pomogło na natychmiastowe lokalizowanie nieprawidłowości w kodzie.

Po ukończenie podstawowej funkcjonalności zdecydowano się na wdrożenie aplikacji na platformie Microsoft Azure i udostępnienie jej publicznie w zamkniętym środowisku dla wybranych użytkowników z możliwością pełnego dostępu do systemu.

Rysunek 39. Analiza czasu odpowiedzi serwera dla ITDesk-Intranet na portalu Azure

Wyniki z analiz i sugestie testerów pozwoliły na wprowadzenie optymalizacji w kodzie źródłowym dla mniejszego obciążenia serwera i szybszego wykonywania zapytań co przełożyło się na lepszy komfort użytkowania aplikacji.

8. Podsumowanie

Realizując założenia przedstawione na wstępie projekt ITDesk został w pełni ukończony uwzględniając wszelkie aspekty. Zastosowane technologie odpowiadają współczesnym standardom tworzenia internetowych aplikacji biznesowych i bezpieczeństwa przetwarzania danych co przekłada się na możliwość natychmiastowego wykorzystania pracy w obecnej formie w środowisku produkcyjnym.

Przejście do fazy testów po zapewnieniu podstawowej funkcjonalności i udostępnienie aplikacji dla docelowej zamkniętej grupy pozwoliło wychwycić i naprawić błędy funkcjonalne oraz wdrożyć nowe zastosowania, a dzięki wykorzystaniu platformy Azure naniesione poprawki mogły być szybko wdrożone i ponownie przetestowane pod kontem poprawnego działania.

Na wybór docelowej platformy w tworzeniu i testowaniu pracy przyczyniła się duża popularność i szeroki zakres dokumentacji wykorzystanej technologii oraz wiedza zdobyta w praktyczny i teoretyczny sposób podczas trwania toku studiów.

Docelowo praca została stworzona w oparciu o obecny system wykorzystywany w firmie świadczącej usługi Helpdesk z wprowadzeniem niezbędnych funkcji w celu optymalizacji czasu pracy spędzanej na uzupełnianiu dokumentacji. Kod aplikacji jest otwarty i pozostawia możliwość dodania kolejnych modułów wprowadzając nową funkcjonalność. Przykładem może być swego rodzaju baza wiedzy FAQ mieszcząca zbiór najczęstszych problemów i rozwiązań.

9. Bibliografia

Źródła bibliograficzne

1. Adam Freeman, „Pro ASP.NET Core MVC: Sixth Edition”, Apress 2016
2. David Sawyer McFarland, „JavaScript i jQuery nieoficjalny podręcznik”, Helion 2012
3. Adam Boduch, „Wstęp do programowania w języku C#” Helion 2006

Źródła internetowe

1. https://docs.microsoft.com/pl-pl/aspnet/core/?view=asp__netcore-5.0 (data odczytu 15.02.2021)
2. https://pl.wikipedia.org/wiki/Model-View-Controller (data odczytu 15.02.2021)
3. https://pl.wikipedia.org/wiki/C_Sharp (data odczytu 15.02.2021)
4. https://docs.microsoft.com/pl-pl/aspnet/core/security/authentication/identity?view=aspnetcore-5.0&tabs=visual-studio (data odczytu 15.02.2021)
5. https://pl.wikipedia.org/wiki/Transact-SQL (data odczytu 16.02.2021)
6. https://www.plukasiewicz.net/Artykuly/Wyrazenia_lambda (data odczytu 16.02.2021)
7. https://pl.wikipedia.org/wiki/Bootstrap_(framework) (data odczytu 17.02.2021)
8. https://developer.snapappointments.com/bootstrap-select (data odczytu 17.02.2021)
9. https://timeago.org/ (data odczytu 17.02.2021)
10. https://pl.wikipedia.org/wiki/Microsoft_Visual_Studio (data odczytu 17.02.2021)

10. Spis rysunków